Verifying Efficacy of Heuristic due to Sharma and Prasad (2003) for the Transportation Problem.

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

February, 2003

By

SUNIL KUMAR MISHRA

DEPARTMENT OF INDUSTRIAL AND MANAGEMENT ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY
KANPUR-208016
INDIA

CERTIFICATE

This is to certify that the present work entitled " Verification of efficacy of heuristic due to Sharma and Prasad (2003) for the Transportation Problem" has been carried out by Mr. Sunil Kumar Mishra under my supervision and that it has not been submitted elsewhere for a degree.

Febraury, 2003

Dr. R. R. K. Sharma
Professor
Dept. of Industrial and Management Engineering
Indian Institute of Technology
Kanpur-208016
India.

Abstract

Sharma and Sharma[6] gives a good dual solution of transportation problem. Prasad[5] used this good dual solution in their heuristic to get a good primal solution for the transportation problem. The present dissertation deals with the verification of the efficacy of the heuristic due to Prasad[5]. The verification is done by measuring the number of iterations taken by the result given by Prasad[5] in Network Simplex Method to reach optimal solution and the number of iterations taken by the result given by Vogel's Approximation Method in Network Simplex Method to reach optimal solution. In our investigation, we found that the number of iterations taken by the result given by Prasad[5] in Network Simplex Method to reach optimal solution is significantly lesser than the number of iterations taken by the result given by Vogel's Approximation Method in Network Simplex Method to reach optimal solution. So it can be suggested that one should now proceed from the good dual solution given by Prasad[5] and reach optimality by the primal method.

ACKNOWLEDGEMENTS

I take this opportunity to express my sincere gratitude towards Dr. R. R. K. Sharma without whose guidance this thesis would not have been in the present form. He was always available in spite of busy schedule. I will always be grateful for him for the interest he has shown in my research.

I would further like to thank all faculty members of the IME Department who were always there with helpful suggestions and provided ample encouragement.

I am indebted to all my IME friends and a special thanks to Amber Pabreja for his time to time kind and valuable helps.

I thank to the staff of IME Department and Hall-IV, for all the help they had extended during my stay at IITK.

I thank all my friends in Hall-IV for making my stay memorable.

Last but not the least I would like to thank almighty God and my family members who made me reach this stage where I could undertake the work of this magnitude.

CONTENTS

Chapter 1

Introduction

Transportation Problems

Transportation Problems are generally concerned with the distribution of a certain product from several sources to numerous localities at minimum cost. Suppose there are m warehouses where a commodity is stocked, and n markets where it is needed. Let the supply available in the warehouses be $a_1, a_2, \ldots, a_m$ and the demands at the markets be $b_1, b_2, \ldots, b_n$. The unit cost of shipping from warehouse i to market j is $\$c_{ij}$. (if a particular warehouse cannot supply a certain market, we set the appropriate c_{ij} at $+\infty$.) we want to find an optimal shipping schedule that minimizes the total cost of transportation from all the warehouses to all the markets.

Standard Formulation of Transportation Problem

We define x_{ij} as the quantity shipped from warehouse i to market j. Since i can assume values from 1,2,….,m and j from 1,2,….n, the number of decision variables is given by the product of m and n. The complete formulation is given below:

Minimize:

$$Z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

Subject to:

$$\sum_{j=1}^{n} x_{ij} = a_i$$ for i = 1, 2, ….,m (supply restriction at warehouse i)

$$\sum_{i=1}^{m} x_{ij} = b_j \quad \text{for } j = 1, 2, \ldots, n \text{ (demand restriction at market j)}$$

$X_{ij} >= 0$ for all (i,j)

The supply constraints guarantee that the total amount shipped from any warehouse does not exceed its capacity. The demand constraints guarantee that the total amount shipped to a market meets the minimum demand at the market.

Finding an initial basic feasible solution

Because of special structure of the transportation problem, there are various distinct methods to find an initial basic feasible solution to start the simplex method without the use of artificial variables.

We shall describe three of the most important ones here.

Northwest Corner rule

This rule generates a feasible solution with no more than (m + n - 1) positive values. The variables that occupy the northwest corner positions in the transportation table are chosen as the basic variables. Thus x_{11} is selected as the first basic variable, and is assigned a value as large as possible consistent with the supply and demand restrictions. We then select the next northwest corner variable as a basic variable, keeping in the view the constraints of supply and demand.

The basic feasible solution obtained by the northwest corner rule may be far from optimal since the transportation costs are completely ignored.

The Least Cost Rule

The only difference between the least cost rule and northwest corner rule is the criterion used for selecting the successive basic variables. In the least cost rule, the variable with the lowest shipping cost will be chosen as the basic variable.
In general the least cost rule provides a better starting solution as compared to the northwest corner rule. But this is not guaranteed in all problems. In fact, examples have been constructed wherein the opposite is true.

Vogel's Approximation Method (VAM)

A proven superior method for finding an initial solution is Vogel's Approximation Method. VAM computes a penalty for each row and column if the smallest cost is not selected from that row or column. VAM defines the penalty as the absolute difference between the smallest and next smallest cost in a given row or column (Note: if two or more cells tie for the minimum cost, the penalty is set at zero). Next, the row or column with the largest penalty is identified and the variable that has the smallest cost in that row or column is selected as the next basic variable (Note: ties may be broken arbitrarily).

Previous Base Work

Sharma & Sharma[6] have given a dual procedure to obtain good solution for the transportation problem. Later Sharma and Prasad[5] have developed a simple Vogel's Approximation Method like procedure to obtain good primal solution to transportation problem by using the solution given by Sharma & Sharma[6]. They also showed that this solution was significantly better than the good solution given by Vogel's Approximation Method.

In this work we wish to test if the journey from good solution given by Sharma & Prasad[5] to optimal solution by network simplex took less number of iterations

compared to iterations taken by network simplex to travel from Vogel's Approximation Method to optimal solution.

We give a brief literature review in Chapter 2.
Our formal problem definition is given in Chapter 4.
Finally in Chapter 5 we give our conclusions.

Chapter 2

Literature Review

1. Introduction

Sharma & Sharma[6] have given a different formulation of transportation problem, which is reproduced below. They use the index i for plants/warehouses and index k for the markets.

1.1 Constants of the problems

The number of markets is K and the number of plants is I. D_k is the demand at market k, and d_k is the demand at market point k as a fraction of total market demand. Hence,

$$d_k = D_k / \sum_{i=1}^{I} D_k \text{ and obviously we have,}$$

$$\sum_{k=1}^{k} d_k = 1$$

C_{ik} is the cost of shipping [$\sum D^K{}_{k=1}$] units of goods from plant number i to market number k. B_i is the supply available at plant i and b_i is the supply available at plant i as a function of total market demand, i.e.

$$b_i = B_i / \sum_{k=1}^{k} D_k \quad \text{and we assume} \quad \sum_{i=1}^{I} b_i = 1$$

They have considered transportation problems in which total supply from all plants is equal to the total demand at all markets.

1.2 Decision variables

X_{ik} is the quantity received as a fraction of total market at market k from plant i.

1.3 Formulation of transportation problem

Problem P:

$$\min \sum_{k} \sum_{i} X_{ik} * C_{ik}$$

$$\text{s.t.} \sum_{k} \sum_{i} X_{ik} = 1, \qquad (1)$$

$$-\sum_{i} X_{ik} >= -d_k$$

for all k=1, ...,K (2)

$$-\sum_{k} X_{ik} >= -b_i$$

for all i= 1,....I (3)

$X_{ik} >= 0$ for all I and k (4)

The problem P is a well-known transportation problem. Approximate heuristics give good starting solution to the transportation problem like Vogel's approximation method (VAM). In this work, they developed a heuristic to obtain a good primal solution to the transportation problem. They gave a brief literature review in section 2 and find that well-known primal approaches solve the problem P in $O(n^3 \log(n))$ time. In section 3 they give a heuristic that obtains a good primal solution (to problem P) by using the dual solution given by Sharma and Sharma[6].

2. Literature Review

Recent works on primal simplex algorithm for minimum cost flow problems have used advanced data structures, see [1]. Uncapacitated transportation problem (problem P) is a special class of minimum cost flow network problems. Ahuja et al.[1] have documented that the uncapacitated minimum cost flow problem can be solved by enhanced capacity scaling algorithm in O(n log(n)S(n,m)) time, where n is the number of nodes and m is the number of areas in the network. For the transportation problem m is $O(n^2)$ and S(n,m) is the running time for solving the shortest path problem with n nodes and m arcs and this is O(m+nlog(n)). Thus enhanced capacity scaling algorithm runs in $O(n^3 \log (n))$ time.

Now we briefly review dual based approaches to solve the minimum cost flow problems. Orlin and Plotkin and Tardos[8] have developed polynomial time dual network simplex algorithms. Algorithm due to Plotkin and Tardos runs in $O(m^3 \log(n))$ time (n is the number of nodes and m is the number of arcs in the network) and is considered more efficient. Ali et al.[9] have noted that dual based approaches take lesser number of pivots to reach optimal solution than the primal based approaches; but computational effort required per pivot is higher for dual based approaches. Hence, Ali et al.[9] have presented efficient implementations of the dual network simplex algorithm for the network flow problem which have resulted in a superior performance. They have also given a reoptimization procedure whereby previous bases are used again to obtain optimal solution for a redefined problem with small changes in the parameters such as capacity or cost. Ali et al.[9] have conducted an extensive experimental study that compared the running times of primal and dual based approaches for the minimum cost network flow problems, and found that dual based approaches perform significantly better.

The approach to uncapacitated transportation problem by Sharma and Sharma[6] is different from approaches given above. They pose the uncapaciatated transportation problem differently (problem P) and find that its dual has a special structure. They exploit this special structure of the dual to develop a computationally attractive $O(cn^2)$ dual based procedure to improve the solution. They refer to this procrdure as heuristic H0. This procedure does not guarantee the optimal solution but empirical investigation revealed that it produced a very good solution. Well-known dual based approaches can use this good solution to get an advanced start while solving the simple transportation problems. However, a good primal solution is not available after heuristic H0 terminates. They develop a heuristic that attempts to give a good primal solution to the transportation problem by using the dual solution given by heuristic H0.

We associate v0, vk and zi as dual variables with constrains 1, 2 and 3, respectively. And write the dual of problem P as follows.

Problem DP:

$$\max \quad v_0 - \sum_{k=1}^{k} d_k v_k - \sum_{i=1}^{I} b_i z_i$$

s.t. $v_0 - v_k - z_i <= C_{ik}$ for all $=1, \ldots, I$

and $k = 1, \ldots, K$, (5)

$v_k >= 0$ for all $k = 1, \ldots, k$. (6)

$z_i >= 0$ for $i = 1, \ldots I$. (7)

v_0 unrestricted sign (8)

Sharma and Sharma[6] have given an $O(cn^2)$ procedure to give a good solution to problem DP The heuristic developed in Section 3 uses this solution to obtain a good primal solution to problem P.

3. Development of the heuristic to obtain a good primal solution for the transportation problem.

At the end of heuristic H0 we have a good solution to problem DP. Let us denote the same by v_0, $v = \{v_k \; \forall k = 1, \ldots, K\}$, $z = \{z_i \; \forall i = 1, \ldots, I\}$. We also have sets I_k, $k=1,\ldots,K$ such that $I_k = \{I: C_{ik} + z_i = C1_k\}$, where $C1_k = \min_i(C_{ik} + z_i)$.

We define slack, $S_{ik} = C_{ik} - v_0 + z_i + v_k$ for every i and k. if $S_{ik}=0$ then associated X_{ik} can take a positive value. For every I_k we can have $X_{ik} \geq 0$ if $i \in I_k$; however if $i \notin I_k$ then complementary slackness conditions demand that $X_{ik}=0$. It is easy to see that if $i \in I_k$, then $X_{ik} > 0$.

Let v_0, $v = \{v_k \; \forall k = 1, \ldots K\}$, $z = \{z_i \; \forall i = 1, \ldots I\}$ be a solution to the fual problem DP as given by a heuristic H0 and $X = \{X_{ik} \; \forall i = 1, \ldots I$ and $k = 1, \ldots K\}$ be a solution to the primal problem P. Then we define a "deviation number" DN associated with the above solutions as follows.

For every arc (i,k) we define the deviation number $DN_{ik} = |C_{ik} - v_0 + v_k + z_i| \; X_{ik}$.

Thus the deviation number DN_{ik} is the product of the absolute value of numbers involved in the complementary slackness property

Then we define $DN = \sum_{ik} DN_{ik}$.

It is easy to see that the lesser the value the value of DN, the lesser the duality gap between primal and dual solutions. For example if DN = 0, then we have an optimal solution as complementary slackness conditions are satisfied.

We have a good solution to the dual problem DP and we wish to construct a good solution to problem P. we have S_{ik} for each cell (i,k). We assign positive X_{ik} 's to these cells in a manner that will keep the associated deviation number as low as possible. We may like to assign X_{ik} 's to those cells (i,k) which have the property that $i \in I_k$. However, as we do not have the optimal dual solution we may have to assign X_{ik} 's to cells (i,k) for which $i \notin I_k$. Assignment to such a cell may have the higher S_{ik} number and may lead to a worse value of DN. Hence, initially we may assign $X_{i1,k1}$ to such a cell (i1,k1) in row i1 that has maximum difference $(S_{i1,k2} - S_{i1,k1})$, where $S_{i1,k1}$ is the least slack and $S_{i1,k2}$ is the next higher slack in a row i1. Later we again assign $X_{i2,k2}$ to such a cell (i2,k2) in row i2 that has maximum difference $(S_{i2,k3} - S_{i2,k2})$, where again $S_{i2,k2}$ is the least slack and $S_{i2,k3}$ is the next higher slack in a row i2; and so on. In fact we obtain maximum difference of next higher slack and least slack after scanning all rows and columns. It is to be noted that slack the heuristic described below is similar to Vogel's approximation method which uses C_{ik} 's instead of S_{ik} 's. Now the details of the heuristic are as given below:

Heuristic H1

Step 0: Let all cells (I,k) be unassigned.

Step 1: Compute

$SK1_k = \min S_{ik} : X_{ik}$

is unassigned $\forall$ cells (i,k)

n_k = {any single I : $SK1_k = S_{ik}$} $\forall$ k

$SK2_k = \min_{i \notin Ik} (S_{ik})$: X_{ik}

is unassigned $\forall$ cells (i,k)

m_i = {any single k : $SI1_i = S_{ik}$} $\forall$ i

$SI2_i = \min_{k \notin mi} (S_{ik})$: X_{ik}

is unassigned $\forall$ cells(i,k).

Step2: Find k:d_k>0 and

$SK2_k - SK1_k >= (SK2_{k1} - SK1_{k1})$ $\forall$ $k1 \neq k$.

Let $D = SK2_k - SK1_k$

Else go to step 6.

Step 3: Find i: $b_i > 0$ and

$(S12i - S11i) >= (S12_{i1} - S11_{i1})$ $\forall$ $i1 \neq i$.

Let $S = S12_i - S11_i$

Step 4: If D>S

Then identify cell (i1, k1) for assignment:

$i1 \in n_{k1}$ and $k1 = k$

Else

Identify cell (i1, k1) for assignment: i1=1

And $k1 \in m_{i1}$

Step 5: Assign $X_{i1k1} = \min(d_{k1}, b_{i1})$

$D_{k1} = d_{k1} - \min(d_{k1}, b_{i1})$.

$b_{i1} = b_{i1} - \min(d_{k1}, b_{i1})$.

Cell (i1,k1) is now labeled as assigned.

Go to step 1.

Step 6: Stop

Result 1. Heuristic H1 runs in $O(n^2)$ time.

Proof. Step 1 of heuristic H1 is executed in O(n) steps and so are steps 2 and 3. They are repeated for a maximum of n times, thus heuristic H1 runs in $O(n^2)$ time.

4. Description of Vogel's approximation method

Now we describe Vogel's approximation method to obtain a good starting solution to the uncapacitated transportation problem. VAM employs an approach similar to heuristic H1 where C_{ik} 's are used instead of S_{ik} 's. We see that $S_{ik} = C_{ik}$ when $v_0 = v_k = z_i = 0$ for all I and k. thus we see that VAM method is applied to a poor dual solution.

Heuristic H2 (VAM): It is identical to heuristic H1 where we use C_{ik} 's instead of S_{ik} 's. and we have a Result 2 that is identical to Result 1.

Result 2. VAM runs in $O(n^2)$ time.

Proof. It is identical to the proof of Result 1.

5. Discussion

Sharma and Prasad[7] showed that their heuristic produced very good primal solution to transportation problem.

We wish to see in this thesis whether solutions given by Sharma and Prasad[7] take lesser no. of iterations when fed to Network Simplex method to reach optimal solution when compared to similar performance by Network Simplex method when fed with a good solution of VAM heuristic.

Chapter3

Research Problem

Sharma and Sharma[6] gives a good dual solution of transportation problem. Sharma and Prasad[7] used this good dual solution in their heuristic and got a good primal solution for the transportation problem. Present work is intended to verify the efficacy of the heuristic due to Sharma and Prasad[7]. The verification is done by measuring the number of iterations taken by the result given by Sharma and Prasad [7] in Network Simplex Method to reach optimal solution and the number of iterations taken by the result given by Vogel's Approximation Method in Network Simplex Method to reach optimal solution.

This required Network Simplex program, which I borrowed from Dr. R.R.K. Sharma (a C program). Input in this Network Simplex program was required to be given from the result of Sharma and Prasad[7], program of which was taken from Saumya Prasad[5], which was in Pascal. But we were unable to use this program because of non-availability of Pascal Compiler in the Institute. So we planned to go ahead by making a new program in C for the same.

The C program for Vogel's Approximation Method was made successfully. The second part was to make a C program for dual solution due to Sharma and Sharma[6] and then give its result as input to a separate C program same as Vogel's Approximation Method but using Sik's in place of Cik's[7]. For this, Pascal program for the dual solution was taken from K.D. Sharma[4] and the whole program of Pascal was converted in C andto do so, I had to learn Pascal from start. But we couldn't successfully run it because the parent Pascal program used "sets" hugely and "sets" are not supported in C programming.

Later we purchased a Pascal compiler for Linux. To use this compiler we installed it on a IME lab Linux PC and in order to access this PC in a remote way, this PC was made a telnet server. But still we were unable to use the available Pascal programs due to

the difference of compiler in which we were attempting to use it and in which it was originally used. The new compiler available with us was not accepting many a things of the program. This led us to change quite a few things in the parent Pascal program, also we added new functions to read and write input(s) and output(s) respectively. Finally this program has been used in this work. We learned about the compiler from the website of Pascal compiler.

Final *modus-operandi* : a seed is given to problem generating program(C program: dsc.c) and also the size of the problem is given to it, i.e. no. of rows and no. of columns This generates a problem of the specified size, i.e. cost matrix, demand and supply are generated, so that sum of all the demands is equal to sum of all the supply. This output is further given to a program(C program: sharma_input.c) which further converts the problem in the structure as required by the Pascal program(saumya.pas) due to Saumya Prasad[5]. The output of this program becomes the input for the saumya.pas . This gives result of both Vogel's Approximation Method and the good primal solution for transportation. The result of Vogel's Approximation Method is given as input in Network Simplex program(C program: tpt_vam.c) and the no. of iterations to reach optimal solution is recorded and the good primal solution is given as input in Network Simplex program(C program: tpt_vamsik.c) and the no. of iterations to reach optimal solution is recorded.

Finally we wish to show that the number of iterations taken by the result given by Sharma and Prasad [7] in Network Simplex Method to reach optimal solution is significantly less than the number of iterations taken by the result given by Vogel's Approximation Method in Network Simplex Method to reach optimal solution.

We conduct t-test on the difference of number of iterations taken by the result given by Sharma and Prasad [7] in Network Simplex Method to reach optimal solution and the number of iterations taken by the result given by Vogel's Approximation Method in Network Simplex Method to reach optimal solution, to prove that former is statistically lesser than the later one.

Chapter 4

Experimental Investigation and Results

We have solved 300 problems, 100 problems each of (50X50) size, (75X75) size and (95X95) size.

The results of 50X50 problems are given in Table 1, 2 & 3.

The results of 75X75 problems are given in Table 4, 5 & 6.

The results of 100X100 problems are given in Table 7, 8 & 9.

Finally Table 10 shows the various t-values.

TABLE 1

PROBLEM SIZE 50X50

S No	Seed	Optimal Solution	VAM(H2)	Dual Solution due to Sharma and Sharma(H0)	Primal Solution due to Sharma & Prasad(H1)
1	3	90280	123988.26	70665.39	81928.5
2	5	56714	79008 13	54319 46	57024 77
3	7	59463	66572.85	49979.25	62138.28
4	17	53238	97595 1	50435 14	57624.93
5	20	67878	89956.54	63709 09	76455 16
6	33	55231	65975 48	53824 12	63969 28
7	40	49265	63738 2	45948 15	58521 99
8	46	55108	76256 23	49925 17	68738.33
9	48	58950	75930.72	55771 35	59918.84
10	49	69532	91408.74	67015 69	71821.11
11	53	63991	74860.42	57600 13	71582.87
12	54	98129	105312.5	67374 21	71342 31
13	56	59127	68716 99	57866 75	61367.63
14	58	52631	81765 7	51090 85	60908 94
15	61	76221	115249.9	74745.33	80740 72
16	62	68793	97869 69	59673 07	76525 12
17	64	58335	87067.33	55493.04	64142.09
18	67	63659	86079 34	60688 26	65600 32
19	68	64351	84786.99	62756.12	67370.79
20	70	66442	118708.68	65790.78	72870.97
21	71	75039	145279 96	72144 57	76243 56
22	72	59921	88410.8	57015 71	60531.89
23	73	63639	75105 91	60152.65	70775.84
24	75	55520	73295 99	51794 43	57104.35
25	77	67019	92378 23	65934 95	71001.72
26	79	51262	80486 67	49273 41	64344.97
27	80	55401	95694 34	54319.4	60609 29
28	81	61231	104168 5	60970.56	64776.29
29	84	54672	78922 89	53847 87	59415.63
30	86	47622	68470 3	45011.47	52839.66
31	88	59743	92168 89	54325 52	65512 65
32	89	104366	161965.07	103515 67	108498.87
33	90	85052	132542 39	83906 23	90791.53
34	91	53160	74635 98	52080.7	66696 32
35	92	53254	76322.9	50735.1	61867.14
36	93	75586	122682.96	68794.53	82291.72
37	94	53185	77905.64	51753.87	56540.84
38	96	54108	82071.89	53561.59	54793 93
39	97	55637	84046.98	54530 49	60284.03
40	98	60912	78047 49	59510.76	62832 84
41	102	70892	147416 98	69568 19	74884 17
42	10	52523	79504.07	51844.86	57529.93
43	104	57928	89861.13	56467 8	59674.12
44	105	48661	80814 36	45841.93	53099.27
45	107	58485	83141.86	50950 03	62187.73

S No	Seed	Optimal Solution	VAM(H2)	Dual Solution due to Sharma and Sharma(H0)	Primal Solution due to Sharma & Prasad(H1)
46	109	46108	75484.97	45436.85	47284.79
47	110	101221	177034 5	99721 18	107237.48
48	111	59558	96035 37	58903 91	62092.69
49	113	63612	99913 71	48713.37	70441.09
50	115	66112	75081.08	65144 03	74626 24
51	116	50890	62882.04	46834 04	62837.18
52	117	62514	110962 27	61796.49	72258.34
53	119	68709	119968 58	67420 74	75946.9
54	120	44327	72030 65	42432 71	54387.23
55	121	55221	89289 19	54806 07	64658.71
56	122	70912	101392 65	69707 99	72729 71
57	124	58559	85663 29	57017 03	68192.69
58	125	64112	90222.73	63301 04	72824 63
59	126	47441	74930 74	46005 07	53830.95
60	127	44517	77517.09	42903 18	51500.63
61	128	52253	87284.48	51229.85	62708.22
62	129	56001	83017 96	55280.68	57151.27
63	131	55109	81176 27	54247 2	64314.26
64	132	50112	60133.93	49400.71	57162.85
65	135	52626	76729 23	51916.64	57050.88
66	136	72123	124759.16	71342.58	76531.1
67	138	46123	76610 78	45481.31	47013.27
68	139	56437	88233.47	55907.82	58814.39
69	140	50012	79799 78	49581 43	54601.78
70	142	61493	86165 31	61155 68	62154.67
71	144	41531	55713 99	40740 79	49969.52
72	146	61993	90526 13	60900.93	64320.77
73	147	53558	81884 03	52432.6	63965.19
74	148	43494	71000 26	42571 74	48279.67
75	149	50661	76339.93	47882.07	58482.84
76	150	44236	77985 98	43139 28	52069.88
77	151	45121	64072 9	43273.7	49074 07
78	156	55006	93828.35	54279 84	57538.62
79	157	46441	104046.04	45628 79	64569.14
80	158	46278	74433 67	45828.82	48715.94
81	159	48223	77150 56	47481.91	50655.51
82	161	53326	90964 69	52270.87	55350.61
83	162	42525	58089 58	41590 84	47031.12
84	163	61929	82511.18	60566 01	77384.73
85	164	53004	82533 45	52326.82	54509.34
86	166	59123	96615 82	58216 64	76740.1
87	167	63232	83071.06	62459 66	70144.9
88	168	59291	81707 03	58106 88	73499.7
89	21	51623	77218 73	50782 98	53458.9
90	22	58401	72702.58	57376.1	65042.97
91	23	52002	71677.21	51375.57	54043 03
92	25	66919	134891 25	65941 14	75634 23
93	28	51236	74637.74	50894 96	52516 02

S No	Seed	Optimal Solution	VAM(H2)	Dual Solution due to Sharma and Sharma(H0)	Primal Solution due to Sharma & Prasad(H1)
94	30	49998	67932.95	49500 32	57462 04
95	32	54987	77698 87	53393 12	73213 81
96	35	53581	75657 62	52379 53	62993.97
97	42	47501	85491 79	46602.76	50777 58
98	43	60633	98466 84	59788 26	62783 65
99	52	50892	78234 17	49426 64	63497.3
100	55	53216	87281 52	52014 87	58981 82

TABLE 2

PROBLEM SIZE 50X50

S No	Seed	No of iterations taken from VAM to Optimal Solution(E1)	No of iterations taken from Primal Solution given by Sharma & Prasad[7] to Optimal Solution(E2)	(E1-E2)X100/E1
1	3	217	152	29.95391705
2	5	271	193	28 78228782
3	7	250	219	12 4
4	17	263	221	15 96958175
5	20	289	264	8 650519031
6	33	283	242	14.48763251
7	40	249	191	23.29317269
8	46	237	189	20 25316456
9	48	352	285	19 03409091
10	49	249	186	25.30120482
11	53	261	227	13.02681992
12	54	278	207	25.53956835
13	56	282	239	15.24822695
14	58	297	221	25.58922559
15	61	272	234	13.97058824
16	62	273	221	19 04761905
17	64	258	203	21.31782946
18	67	323	270	16.40866873
19	68	279	237	15 05376344
20	70	283	239	15 54770318
21	71	266	201	24.43609023
22	72	261	199	23.75478927
23	73	292	258	11 64383562
24	75	296	244	17.56756757
25	77	288	236	18 05555556
26	79	272	229	15.80882353
27	80	281	225	19.92882562
28	81	327	252	22.93577982
29	84	269	209	22.30483271
30	86	302	247	18.21192053
31	88	263	212	19 39163498
32	89	299	219	26 75585284
33	90	316	258	18 35443038
34	91	288	251	12 84722222
35	92	279	240	13.97849462
36	93	292	247	15.4109589
37	94	268	223	16 79104478
38	96	273	242	11.35531136
39	97	309	256	17.15210356
40	98	273	221	19 04761905
41	102	263	204	22 43346008
42	10	291	235	19.24398625
43	104	266	198	25.56390977
44	105	279	232	16.84587814
45	107	281	219	22 06405694
46	109	276	208	24.63768116

S No	Seed	No. of iterations taken from VAM to Optimal Solution(E1)	No. of iterations taken from Primal Solution given by Sharma & Prasad[7] to Optimal Solution(E2)	(E1-E2)X100/E1
48	111	283	248	12.36749117
49	113	277	230	16 96750903
50	115	322	298	7.453416149
51	116	267	245	8 239700375
52	117	319	252	21.0031348
53	119	312	231	25.96153846
54	120	256	233	8.984375
55	121	261	229	12 2605364
56	122	296	223	24 66216216
57	124	289	256	11 41868512
58	125	302	263	12.91390728
59	126	267	218	18 35205993
60	127	273	222	18 68131868
61	128	283	231	18.3745583
62	129	309	267	13.59223301
63	131	282	248	12.05673759
64	132	254	237	6.692913386
65	135	295	226	23.38983051
66	136	316	249	21.20253165
67	138	296	209	29.39189189
68	139	283	217	23.32155477
69	140	299	264	11.70568562
70	142	278	243	12.58992806
71	144	289	254	12.11072664
72	146	298	237	20.46979866
73	147	274	226	17.51824818
74	148	261	202	22.60536398
75	149	279	238	14.6953405
76	150	280	233	16.78571429
77	151	301	269	10.63122924
78	156	275	228	17.09090909
79	157	296	241	18 58108108
80	158	279	249	10.75268817
81	159	263	210	20 15209125
82	161	312	268	14.1025641
83	162	296	231	21.95945946
84	163	276	259	6 15942029
85	164	305	246	19 3442623
86	166	287	239	16 72473868
87	167	261	203	22 22222222
88	168	283	231	18.3745583
89	21	300	253	15 66666667
90	22	289	262	9.342560554
91	23	292	259	11.30136986
92	25	308	251	18 50649351
93	28	289	216	25.25951557
94	30	276	243	11.95652174
95	32	288	263	8 680555556
96	35	253	211	16 60079051

S No	Seed	No of iterations taken from VAM to Optimal Solution(E1)	No. of iterations taken from Primal Solution given by Sharma & Prasad[7] to Optimal Solution(E2)	(E1-E2)X100/E1
98	43	299	228	23.7458194
99	52	276	258	6 52173913
100	55	291	236	18.90034364

TABLE 3

PROBLEM SIZE 50X50

S.No	(H2-H0)x100/H0	(H1-H0)x100/H0
1	75 45825474	15 93865116
2	45 45087525	4 980369834
3	33 20097841	24 32815618
4	93.50615464	14.25551709
5	41.19890898	20.00667409
6	22 57604955	18.84872433
7	38.71766328	27.36528021
8	52 74105226	37 68271595
9	36.1464623	7 43659603
10	36 39901343	7.170589454
11	29 96571362	24 27553549
12	56 30981053	5 889642342
13	18 75038774	6.049899122
14	60 03981143	19 21692436
15	54 19010124	8 021089746
16	64 00981213	28 24062848
17	56 8977479	15 58582842
18	41 83853681	8 093921295
19	35 10553234	7 353338607
20	80.43361091	10 76167512
21	101 3733813	5 68163342
22	55 06392887	6 16703712
23	24 85885493	17.6603857
24	41 51326697	10.25191319
25	40 10510359	7.68449813
26	63 34706691	30 58761308
27	76 16972942	11 57945412
28	70 85048915	6 24191413
29	46.56641015	10 33979617
30	52 11744917	17.39154487
31	69 66039165	20 59277113
32	56 46430149	4 813957153
33	57.96489724	8.20594609
34	43 30832727	28 06340929
35	50 4341176	21 94149612
36	78 33243428	19 61956859
37	50.53104241	9.249491874
38	53 22900235	2 300790548
39	54 12841513	10.55105135
40	31 14853516	5 582318223
41	111.9028539	7.641394724
42	53 349956	10.9655422
43	59 13694176	5 678138691
44	76 28917456	15 831227
45	63 18314238	22 05631675
46	66.13160904	4.067051303
47	77 52948772	7.537315543
48	63 03734336	5.413528576

S.No	(H2-H0)x100/H0	(H1-H0)x100/H0
49	105 105313	44.60319621
50	15 25396878	14 55576206
51	34 26567514	34 16989011
52	79 56079706	16 9295214
53	77 94017093	12 64619759
54	69.75265073	28.17288832
55	62.91843221	17 97727879
56	45 45341216	4 334825893
57	50 24158572	19 60056495
58	42.52961721	15 04491869
59	62.87496139	17 01090771
60	80 67912448	20 03919057
61	70.3781682	22 40562875
62	50 17535964	3 383804251
63	49 64140085	18 55775045
64	21 72685372	15 71260818
65	47 7931353	9 889391918
66	74 87335053	7 272683438
67	68 44453249	3 368328661
68	57 81955011	5 198861268
69	60 94691097	10 12546431
70	40 89502398	1 633519568
71	36.75235556	22 65230988
72	48 64490575	5 615415068
73	56 17007358	21 99507558
74	66 77791417	13 40779118
75	59 43322835	22 13933107
76	80 77719424	20 70178269
77	48 06429771	13 40391508
78	72 86040268	6 003665449
79	128 0271732	41 50964775
80	62 41672816	6 299791267
81	62 48411237	6.683808634
82	74 02559016	5 891885863
83	39 66916754	13.08047637
84	36 23347485	27.76923888
85	57 72685976	4.170939491
86	65 9591141	31.81815371
87	32 99953922	12.3043257
88	40 61507002	26 49052918
89	52 05631887	5 269324486
90	26.71230704	13.3624802
91	39.51613578	5 192078647
92	104 5631149	14 69960938
93	46 65055243	3 185109095
94	37 23739564	16.08417885
95	45 52225081	37 12217979
96	44.44119678	20 26448118
97	83.44791167	8.958310624
98	64 69260019	5.009996946
99	58 28340749	28.46776556
100	67 80109226	13.39415056

TABLE 4

PROBLEM SIZE 75X75

S No	Seed	Optimal Solution	VAM(H2)	Dual Solution due to Sharma and Sharma(H0)	Primal Solution due to Sharma & Prasad(H1)
1	1	52274	82947.95	50748.91	75337.57
2	2	69186	97394 53	67760.23	84247.92
3	5	73051	105938.67	71424.92	85993 84
4	9	53281	85326 36	52526 9	70188.98
5	10	53201	89201.4	52073 2	68281 78
6	11	59222	92816 86	58654.39	63391.16
7	13	55523	91931 93	53723.54	60503 38
8	14	61107	104092 48	59943 19	69783 45
9	15	67513	102300 81	66210 03	90258.54
10	16	51407	79498.29	50460 04	56014.76
11	17	52116	81876.34	51397.97	58232.52
12	18	56101	81055 23	55168.97	66945.85
13	25	53112	87617 24	51828.23	59771.92
14	28	57636	84711.79	56774 67	61494.52
15	31	46352	75878 1	45115 17	48865.04
16	37	67319	105306 97	65976.53	82723.8
17	40	82005	140454 66	81181 39	89757.22
18	42	53539	84238 42	52740.44	61322.31
19	43	67396	136683.71	66054 63	76965 77
20	45	64883	92467 96	63614.1	72318.67
21	48	53801	92241.24	52728.76	63092.5
22	51	53182	70529 8	51942 64	57175 24
23	57	57396	85559.56	56059 41	92041 23
24	61	75112	133174 93	74250 51	80325.38
25	63	72109	135937.71	69891 4	79537.19
26	64	55012	98467.88	54274 22	69844
27	65	71272	120393.67	70234.11	85865 92
28	66	65672	116341.67	64674.01	71345.01
29	72	62610	92295 27	61009 1	71702 78
30	73	59602	74326 23	58557 55	68636 82
31	75	59341	68884.66	48129.92	63605.9
32	76	51039	70342 01	59644.57	64684.46
33	78	62311	107597 72	61285 99	63691 76
34	79	60432	87497.74	57688 61	63738 95
35	80	71916	105900.06	70972.91	78247.64
36	82	54557	87961 26	53213.26	59506 47
37	83	52921	75589 5	51312 68	60021.78
38	84	54663	85842 93	53461.31	59881.46
39	85	56994	97684 59	55614 2	63639 64
40	87	73820	133174 98	72419 07	83868.57
41	89	54107	67111.68	52257.17	59565.44
42	90	79887	120251.06	78302.93	82476 56
43	92	57921	90174 16	56350 6	67811 82
44	94	55908	86989 05	54816.9	72027.79
45	95	58219	100863 4	55887.01	70356 67
46	97	57422	84324 46	56307 79	66921 13
47	107	54998	98962.27	53704 86	61301 95

S No	Seed	Optimal Solution	VAM(H2)	Dual Solution due to Sharma and Sharma(H0)	Primal Solution due to Sharma & Prasad(H1)
48	109	75369	148092 86	74133 51	85418 33
49	112	60758	85425 42	59297 82	69730 87
50	114	65107	115172.47	64303 08	67305 72
51	115	53256	80155.74	52057.21	71696 25
52	120	61833	87510 45	60712.66	64487.84
53	123	611009	88508 57	59054 46	78426 26
54	124	63630	86745.54	62790 92	70370 14
55	125	67119	122130.74	66484.03	76379.42
56	126	59623	97812 86	57628 15	65626.4
57	127	46814	82558.17	45588.86	53586 51
58	128	67352	104830 26	66438.3	76275.51
59	136	69119	131378 52	67931 61	74968.75
60	137	49153	78145 03	48005.55	56173 45
61	143	52883	82602 82	51609 82	54983.9
62	146	52752	86260	51592.3	58554 19
63	147	57138	94382.15	55308.09	64225.45
64	148	57542	97941 81	56530.04	62513 38
65	149	50773	82649 32	49812.32	56205.87
66	151	60952	98825 86	59880 11	64401.29
67	152	45308	72689 97	44247 33	53514 26
68	153	53884	96337 14	52334 02	58463
69	154	53776	83617.98	52726 37	59084.02
70	156	52812	94211 85	51225 67	58310 6
71	157	57633	78506 89	56594 28	69143.48
72	158	62882	93909 3	61840 99	69053.04
73	159	50176	64388 62	47527 22	56939.85
74	161	63529	103558.05	62417.97	67467.72
75	165	70012	150308.14	68659 95	86096 86
76	166	57868	82487 56	56543 72	66392.57
77	167	68998	124707.98	67560.79	75372.94
78	170	66108	100954.36	64925.51	88954.33
79	171	54881	79126.26	54035 88	60170.89
80	173	39999	68649 15	39450.77	51973.06
81	175	81872	152033.49	80622 73	104680.05
82	176	59709	111650.06	58966 12	71387.77
83	178	57234	82841 19	56184 14	80427.24
84	179	60692	81729.63	58931 97	80793.17
85	181	46881	76030 98	45348.28	57161.31
86	182	49124	91831.57	48044.05	63943 44
87	183	54252	71901.45	53097.49	68939 57
88	184	53439	85375.58	51027 28	64386.49
89	185	69577	129577 32	66065 24	77896 65
90	186	75783	154540 84	74255 27	95108.35
91	187	52934	78182.92	51720 38	59981 72
92	188	52663	66111 37	51281 12	61324.35
93	189	63392	94268.63	62575 4	69986 89
94	190	57252	87028 03	56425.14	68392 54
95	194	50886	82595 66	49603 05	62320 89
96	196	55451	70038.76	54327 83	58864.31

S No	Seed	Optimal Solution	VAM(H2)	Dual Solution due to Sharma and Sharma(H0)	Primal Solution due to Sharma & Prasad(H1)
97	200	59758	98790 15	58213 94	63334 53
98	201	59992	89662 28	59095 03	61823.57
99	205	65760	105408 18	64863 6	69602.43
100	207	52889	96724 95	51966.57	63368.81

TABLE 5

PROBLEM SIZE 75x75

S No	Seed	No. of iterations taken from VAM to Optimal Solution(E1)	No of iterations taken from Primal Solution given by Sharma & Prasad[7] to Optimal Solution(E2)	(E1-E2)X100/E1
1	1	490	456	6 93877551
2	2	502	459	8 565737052
3	5	479	441	7.933194154
4	9	461	417	9.544468547
5	10	454	410	9.691629956
6	11	484	418	13.63636364
7	13	476	429	9 87394958
8	14	512	446	12.890625
9	15	508	466	8.267716535
10	16	463	419	9.503239741
11	17	484	439	9 297520661
12	18	509	461	9 430255403
13	25	513	446	13.06042885
14	28	477	436	8 595387841
15	31	481	459	4 573804574
16	37	489	422	13 70143149
17	40	522	459	12 06896552
18	42	511	462	9 589041096
19	43	522	457	12 45210728
20	45	456	416	8.771929825
21	48	478	441	7.740585774
22	51	486	449	7.613168724
23	57	459	418	8.932461874
24	61	507	464	8.481262327
25	63	513	453	11.69590643
26	64	580	539	7 068965517
27	65	502	482	3 984063745
28	66	513	467	8.966861598
29	72	477	436	8.595387841
30	73	449	409	8 908685969
31	75	451	423	6.208425721
32	76	457	416	8 971553611
33	78	504	452	10 31746032
34	79	476	439	7.773109244
35	80	518	457	11.77606178
36	82	486	429	11.72839506
37	83	476	431	9.453781513
38	84	451	399	11.52993348
39	85	463	416	10.1511879
40	87	516	423	18.02325581
41	89	498	449	9.83935743
42	90	501	419	16.36726547
43	92	493	449	8 92494929
44	94	452	411	9.07079646
45	95	497	451	9.255533199
46	97	481	433	9 979209979
47	107	508	462	9.05511811

S No	Seed	No. of iterations taken from VAM to Optimal Solution(E1)	No. of iterations taken from Primal Solution given by Sharma & Prasad[7] to Optimal Solution(E2)	(E1-E2)X100/E1
48	109	523	471	9.942638623
49	112	496	447	9.879032258
50	114	522	410	21.4559387
51	115	476	430	9.663865546
52	120	465	407	12.47311828
53	123	453	416	8.167770419
54	124	489	436	10.83844581
55	125	522	432	17.24137931
56	126	497	437	12 07243461
57	127	463	419	9.503239741
58	128	508	433	14 76377953
59	136	528	436	17 42424242
60	137	480	429	10.625
61	143	471	416	11 67728238
62	146	488	410	15 98360656
63	147	467	428	8 35117773
64	148	472	421	10 80508475
65	149	489	436	10.83844581
66	151	481	447	7 068607069
67	152	475	419	11 78947368
68	153	470	411	12.55319149
69	154	497	423	14.88933602
70	156	489	419	14.31492843
71	157	477	423	11.32075472
72	158	462	416	9 956709957
73	159	439	401	8.656036446
74	161	526	429	18.44106464
75	165	547	439	19.7440585
76	166	470	432	8 085106383
77	167	530	437	17.54716981
78	170	489	450	7 975460123
79	171	467	431	7 708779443
80	173	484	446	7.851239669
81	175	526	428	18.63117871
82	176	519	433	16.57032755
83	178	479	451	5.845511482
84	179	488	449	7 991803279
85	181	478	436	8.786610879
86	182	463	424	8.423326134
87	183	452	433	4 203539823
88	184	492	447	9.146341463
89	185	507	438	13.60946746
90	186	512	450	12.109375
91	187	469	423	9.808102345
92	188	448	416	7 142857143
93	189	463	422	8.855291577
94	190	470	428	8.936170213
95	194	482	426	11.61825726
96	196	457	410	10 28446389

S No	Seed	No. of iterations taken from VAM to Optimal Solution(E1)	No of iterations taken from Primal Solution given by Sharma & Prasad[7] to Optimal Solution(E2)	(E1-E2)X100/E1
97	200	483	437	9.523809524
98	201	472	411	12.92372881
99	205	499	418	16 23246493
100	207	486	434	10.69958848

TABLE 6

'ROBLEM SIZE 75X75

S.No.	(H2-H0)x100/H0	(H1-H0)x100/H0
1	63.44774696	48 45160221
2	43 73406052	24 3323997
3	48.32172021	20 39753072
4	62.44316722	33 62482842
5	71.30001613	31.12652958
6	58.2436711	8.075729711
7	71.12038782	12 61986831
8	73.65188606	16 41597653
9	54.50953579	36 32155128
10	57.54702137	11.00815616
11	59.29878164	13 29731505
12	46.92177505	21 34692745
13	69.05312028	15.32695599
14	49.20701433	8.313302393
15	68.18755199	8.31177185
16	59 61277442	25 3836781
17	73.01337166	10 56378808
18	59.72263409	16.27189686
19	106 9252526	16.51835761
20	45 3576487	13 68339723
21	74 93534838	19 65481456
22	35 78401098	10 07380449
23	52.62301191	64 18515643
24	79.35894312	8 181586901
25	94 49847907	13 80111144
26	81.4266147	28 68724783
27	71.41766301	22 25672113
28	79 88937133	10 31480807
29	51 28115314	17.52800812
30	26 92851733	17.2125883
31	43.12232391	32.15459323
32	17.93531247	8.449872302
33	75.56658545	3.925481174
34	51.67247053	10.48792821
35	49 2119458	10 25000948
36	65.29951369	11 8263944
37	47.31154171	16.97260794
38	60 57019553	12 00896499
39	75 64684919	14 4305591
40	83 89490503	15.81006218
41	28 42578349	13 9852005
42	53 57159687	5.33010706
43	60.02342477	20.33912682
44	58.6902032	31.39705091
45	80.47735959	25.89091812
46	49.75629482	18.84879517
47	84.27060419	14.14600094
48	99.7650725	15 22229286
49	44 06165353	17.59432303

S.No.	(H2-H0)x100/H0	(H1-H0)x100/H0
50	79 1087923	4 669511943
51	53.97625036	37 72587889
52	44 13871835	6.218110028
53	49 87618209	32 80328023
54	38 14981529	12 07056689
55	83.69936359	14 88386008
56	69.7310429	13 87906778
57	81.09285909	17.54299186
58	57.78588555	14 80653478
59	93.39821329	10 35915386
60	62.78332401	17.01449103
61	60 05252489	6.537670544
62	67.19549235	13 49404853
63	70.64800104	16.12306626
64	73.25621917	10 5843548
65	65 92144273	12 8352785
66	65 03954318	7 550386931
67	64 28103119	20.9434784
68	84 08129167	11 71127309
69	58.58853928	12 05781851
70	83.91531043	13 83081959
71	38 71877158	22 17397235
72	51.85607475	11.66224862
73	35 47735382	19.80471401
74	65.91063439	8.09021825
75	118 916763	25.39604238
76	45 88279653	17.41811469
77	84 58632589	11 56314188
78	55.49259451	37.00982865
79	46.43281464	11.3535858
80	74 01219292	31.74156043
81	88.57397908	29 83937656
82	89.34611943	21.0657408
83	47.44586284	43 14936564
84	38.68470713	37.09565453
85	67.66011853	26 04956572
86	91.14035973	33 09335912
87	35.41402805	29.83583593
88	67 31360167	26 1805254
89	96.13539586	17 90867633
90	108 1210398	28.082963
91	51 16462795	15.9730845
92	28 91951268	19.58465416
93	50.64806617	11.84409528
94	54.23626773	21.20934038
95	66 51326884	25.63922985
96	28.91875122	8.350195471
97	69 70187897	8.796157759
98	51.72558504	4 617207234
99	62.5074464	7 305838714
100	86.12917882	21.94149046

TABLE 7

'ROBLEM SIZE 95X95

S No	Seed	Optimal Solution	VAM(H2)	Dual Solution due to Sharma and Sharma(H0)	Primal Solution due to Sharma & Prasad(H1)
1	2	66753	134894 73	63420 04	72712.75
2	3	49387	73736 85	47871 4	55149.53
3	4	60505	96366.47	59049.11	66776.4
4	7	54717	92626.21	50319 6	73940.97
5	9	56975	100485 86	56592.99	63525.24
6	12	64131	98815 83	61008.22	78250.93
7	14	64784	134151 88	63565.51	69313.59
8	20	59083	97155 18	56826.87	65627.76
9	26	55408	131836 92	53021.61	76413.48
10	27	65542	131698 09	59657.51	68752.33
11	28	51912	79825.05	50938 87	62376.47
12	29	63301	84656.37	52360 16	68856.78
13	31	57042	81850.57	50582 17	58677.08
14	32	58293	91913 78	56246 63	71604.32
15	33	74495	142884 15	73629.15	91688 71
16	35	56272	73278 19	53568 33	66850.95
17	42	57852	91576.65	54662.36	71664 46
18	46	69543	143874 1	68457 17	77071.71
19	47	56288	87366 59	55134.88	72208 54
20	48	96428	211071 04	90060 74	115433.84
21	49	81704	142992 13	69734 15	79235 18
22	50	57702	74513.21	51829 98	69739.86
23	56	62966	93236 77	60602 86	68842.78
24	57	57561	87188.34	56731.49	70064.17
25	58	59062	90249 35	54399 55	70919.92
26	59	59923	87471 56	57582 47	61601.98
27	63	72983	131553.76	64587 01	75296.85
28	64	55615	92480.37	53085.71	58557.84
29	68	56745	123490.8	55701.93	72527.41
30	73	59518	108487 89	56682 43	64017.44
31	74	77991	153281 86	71000 79	79426 56
32	79	55181	79272 31	48774 34	59640.91
33	80	68723	102371 88	62885 18	71884.02
34	81	57989	80097 65	54489.53	64091.92
35	84	51107	76156 43	48891.49	66407.99
36	88	58145	77499 43	48358.3	58217.89
37	90	75112	161202 65	73626.66	86807 2
38	93	81136	149566 47	80688	118932 73
39	94	68242	140082 77	59534.79	73163.82
40	95	58898	86690.74	56093.54	62801.25
41	98	67884	98009 05	62724 47	75139 92
42	103	61849	90654 77	60357 7	77234.77
43	104	66636	85708.87	61306.82	75888 1
44	107	57912	78491 03	53458 74	59558.64
45	110	58083	92696.42	56134.87	73627 59

S No	Seed	Optimal Solution	VAM(H2)	Dual Solution due to Sharma and Sharma(H0)	Primal Solution due to Sharma & Prasad(H1)
46	115	61820	85188.21	59266 66	75768.33
47	116	62128	82035.29	59617 89	71881 19
48	120	58870	80021.14	52888 16	68722 86
49	121	62970	87069.51	56317.17	71156.25
50	123	49659	70584.01	43915.85	52566 07
51	126	63876	79292 9	57845 93	68283 92
52	127	65453	113401 55	63002 55	89145.02
53	128	63144	122049.13	58778.63	72057 71
54	132	63719	111738 35	61725 43	70887.99
55	135	61209	100481.24	58628 89	61451 66
56	136	73347	101047 44	66057 52	75883.95
57	141	70442	102822.41	65810.03	88008.83
58	144	54353	92616 97	52441 21	59693.16
59	145	62820	98664 35	62010.91	78514.4
60	146	58987	99003 56	58207 83	77601 77
61	147	60078	78973 24	55702 75	65896 42
62	148	77146	125946 42	69618	89304.34
63	151	57791	97855 84	55836 34	65186.24
64	156	59802	91563 25	56232 8	73714 34
65	160	50510	78516.99	48644 31	64428.26
66	161	60641	95040 29	56746.46	71367.38
67	162	63159	104256.51	59571.83	67491.12
68	165	52198	71116 84	48556.86	62782 49
69	166	62489	83356.9	55657.68	67290.33
70	170	69082	102299 14	64160 28	83108.74
71	176	56932	92692 22	54623.79	63015 51
72	179	59426	82821 03	57934.39	65566.43
73	181	44547	72168.93	42477.12	62892 23
74	185	60869	120786 19	57768.4	79385.01
75	187	70122	108658.73	66664 7	83291.96
76	188	53803	76778 02	49242 52	56441 39
77	192	59320	80921 8	57219 17	71833 57
78	197	56104	77552 5	52937 02	63822
79	198	61589	85000 01	58482.68	74131.8
80	200	58166	80145 19	57369 67	67449.04
81	201	74792	162168 82	73890 86	89851 55
82	207	60313	100307 88	57192 28	62721 71
83	208	61563	80414.7	59260 93	69604.72
84	211	56255	74017.84	53897.82	62288 68
85	215	62870	103445 07	57653 06	66035 36
86	222	62888	78132.37	59360.98	70287.14
87	226	62848	87251 57	58741.07	70176.67
88	228	62837	99962 19	57080.13	66223.42
89	232	52580	88540.05	48046.64	72321.44
90	235	85690	149323 87	75506.66	92244.48
91	236	68484	92811 98	62323.69	78637 75
92	240	65476	127891.84	61866 98	68985.49
93	243	56078	73738 41	52389 71	61942.25
94	249	57436	95843.01	55865 6	75467 79

S No	Seed	Optimal Solution	VAM(H2)	Dual Solution due to Sharma and Sharma(H0)	Primal Solution due to Sharma & Prasad(H1)
95	252	59716	95472 32	57606 21	63007 46
96	253	52703	89799 9	50113 09	65836.83
97	255	68288	91147.72	63707 88	74223 3
98	263	55963	77419 3	53691	62222.83
99	267	56422	86476.58	48748 74	66250 02
100	273	53672	100209.52	52845.93	61073.52

TABLE 8

PROBLEM SIZE 95X95

S No	Seed	No. of iterations taken from VAM to Optimal Solution(E1)	No of iterations taken from Primal Solution given by Sharma & Prasad[7] to Optimal Solution(E2)	(E1-E2)X100/E1
1	2	667	652	2.248876
2	3	687	677	1.455604
3	4	750	681	9.2
4	7	690	661	4.202899
5	9	724	697	3 729282
6	12	640	599	6.40625
7	14	674	632	6.231454
8	20	686	649	5.393586
9	26	721	682	5.409154
10	27	713	676	5 189341
11	28	751	720	4.12783
12	29	645	602	6.666667
13	31	670	645	3.731343
14	32	723	699	3.319502
15	33	717	697	2.7894
16	35	756	724	4.232804
17	42	749	722	3.604806
18	46	761	732	3.810775
19	47	658	637	3 191489
20	48	691	664	3.907381
21	49	708	671	5 225989
22	50	721	637	11.65049
23	56	718	690	3.899721
24	57	766	726	5.221932
25	58	691	654	5.354559
26	59	703	667	5.12091
27	63	705	679	3.687943
28	64	697	659	5.451937
29	68	733	706	3.683492
30	73	678	618	8.849558
31	74	731	706	3 419973
32	79	746	724	2.949062
33	80	665	621	6.616541
34	81	706	663	6.090652
35	84	711	642	9.704641
36	88	669	609	8.96861
37	90	727	698	3 988996
38	93	729	671	7.956104
39	94	727	684	5.914718
40	95	686	679	1.020408
41	98	644	608	5.590062
42	103	699	689	1.430615
43	104	646	612	5.263158
44	107	761	696	8.541393
45	110	772	730	5.440415

S No	Seed	No. of iterations taken from VAM to Optimal Solution(E1)	No. of iterations taken from Primal Solution given by Sharma & Prasad[7] to Optimal Solution(E2)	(E1-E2)X100/E1
46	115	697	630	9.612626
47	116	706	642	9.065156
48	120	692	625	9.682081
49	121	701	666	4.992867
50	123	701	638	8.987161
51	126	697	659	5 451937
52	127	712	666	6.460674
53	128	696	631	9.33908
54	132	712	637	10.53371
55	135	703	612	12.94452
56	136	714	659	7.703081
57	141	689	641	6.966618
58	144	763	672	11.92661
59	145	708	688	2 824859
60	146	732	681	6.967213
61	147	709	668	5 782793
62	148	690	666	3.478261
63	151	742	690	7.008086
64	156	699	657	6.008584
65	160	687	651	5.240175
66	161	752	709	5.718085
67	162	779	690	11 4249
68	165	748	704	5.882353
69	166	703	655	6.827881
70	170	768	728	5.208333
71	176	705	649	7.943262
72	179	695	639	8 057554
73	181	682	666	2.346041
74	185	760	684	10
75	187	683	637	6.734993
76	188	701	684	2.425107
77	192	773	733	5.174644
78	197	683	625	8.491947
79	198	712	663	6.882022
80	200	715	685	4 195804
81	201	684	637	6.871345
82	207	782	723	7 544757
83	208	722	659	8.725762
84	211	739	687	7.036536
85	215	783	734	6.257982
86	222	694	628	9 510086
87	226	718	652	9.192201
88	228	782	708	9.462916
89	232	728	659	9 478022
90	235	666	612	8.108108
91	236	752	697	7 31383
92	240	787	711	9.656925
93	243	711	651	8.438819
94	249	767	700	8.735332

S No	Seed	No. of iterations taken from VAM to Optimal Solution(E1)	No. of iterations taken from Primal Solution given by Sharma & Prasad[7] to Optimal Solution(E2)	(E1-E2)X100/E1
96	253	651	604	7.219662
97	255	712	629	11.6573
98	263	788	736	6.598985
99	267	748	670	10.42781
100	273	704	651	7.528409

TABLE9

PROBLEM SIZE 95X95

S.No.	(H2-H0)X100/H0	(H1-H0)X100/H0
1	112.7004808	14.65263976
2	54 03111252	15.20350355
3	63 19715911	13 08620909
4	84.07580744	46 94268237
5	77.55884607	12 24930862
6	61.97133763	28.26292916
7	111.0450777	9.042765487
8	70.9669739	15.48719822
9	148 6475232	44.11761544
10	120 756934	15.24505465
11	56.70753984	22.45358014
12	61 68088486	31.50605346
13	61.81703948	16.00348502
14	63.41206575	27.30419582
15	94 05921432	24.5277312
16	36 79386682	24.79565818
17	67.53146041	31 10385282
18	110 1665903	12.58383892
19	58 45974454	30.96707565
20	134 3652073	28.17331947
21	105.0532343	13.62464445
22	43.76468986	34.55505867
23	53.84879525	13.59658604
24	53 68596876	23.50137463
25	65 90091278	30.36857842
26	51 90657851	6 980440401
27	103 684549	16.58203407
28	74.20953775	10.30810363
29	121.6993199	30.20627831
30	91.39597579	12.94053554
31	115.8875415	11.86714965
32	62 52871899	22.27927636
33	62.79174203	14.30995347
34	46.99640463	17.62244967
35	55.76622844	35.82729837
36	60.26086525	20.38861995
37	118.9460312	17.90185783
38	85 36395747	47.39828723
39	135.2956481	22.89254737
40	54.5467446	11 95807931
41	56.25329317	19.79363078
42	50 19586565	27 96175136
43	39 8031573	23.78410754
44	46 82543958	11 41048218
45	65 13161962	31.16194978
46	46 7765443	27.84309087
47	37.6018004	20.56983231
48	51.30255997	29 93997144
49	54.60562028	26.3491223

S.No.	(H2-H0)X100/H0	(H1-H0)X100/H0
50	60.72559224	19.69726192
51	37 07602246	18.04446743
52	79.9951748	41.4943046
53	107 6420121	22 59168
54	81.02482235	14 84406022
55	71.38519934	4 814640018
56	52.96886713	14 87556602
57	56 24124469	33.73163635
58	76 61104692	13.82872363
59	59.1080505	26.61384908
60	70 08632687	33 31843843
61	41 77619597	18.30011983
62	80.91071275	28.27765808
63	75.25475345	16.74518781
64	62.82890057	31.08779929
65	61.41043012	32 4476799
66	67.48232401	25.7653429
67	75.00974874	13 2936826
68	46.4609532	29 2968491
69	49 7671121	20.90035014
70	59.44310093	29 53300703
71	69 69203345	15.36275678
72	42 95659279	13.17359171
73	69 90071361	48.06142695
74	109 0869576	37.41943692
75	62 99290329	24.94162578
76	55.91813741	14.61921526
77	41.42428141	25 54109051
78	46 4995574	20.56213213
79	45 34219362	26.75855484
80	39.69958342	17 56916154
81	119.4707437	21 60035761
82	75.3870977	9.668140525
83	35.69598047	17.45465351
84	37.32993282	15 56808791
85	79.42685089	14.53921093
86	31.62243952	18.4062999
87	48.53588809	19 46781017
88	75.12607277	16 01834123
89	84.27937937	50.52340809
90	97.76251525	22.16734259
91	48 91926328	26.1763384
92	106 7206772	11 50615401
93	40.7497961	18.23361878
94	71.55997609	35 08812221
95	65.7326875	9.376159272
96	79.19449788	31.3765126
97	43.07134376	16.50568187
98	44.19418525	15.89061481
99	77.39244132	35.90098944
100	89.62580467	15.56901355

Table 10

Problem Size	t-value for difference between [(H2-H0)X100/H0] and [(H1-H0)X100/H0]	t-value for the differnce between E1 & E2
50X50	18 74	31.06746
75X75	17.96	31.46821
95X95	17.12	18.52667

t-value at 0 0001 significance level = 3.373

So, it can be concluded that all 't' values in Table 10 are significant.

This means that heuristic H0 which is used to produce good starting primal solution reduces the effort taken by "Network Simplex Algorithm" for the Transportation Problem to reach Optimal Solution.

Chapter 5

Conclusions and Future Research Direction

In this work we have found that the number of iterations taken by the result given by Sharma and Prasad [7] in Network Simplex Method to reach optimal solution is significantly lesser than the number of iterations taken by the result given by Vogel's Approximation Method in Network Simplex Method to reach optimal solution.

This has enhanced the performance of the heuristic given by Sharma and Prasad [7], and is an important result which saves lots of computation effort and time.

It is suggested that one should now proceed from the good dual solution given by Sharma and Sharma [6] and reach optimality by the primal method[7]. A future research topic would be to find the efficacy of this dual based approach.

REFERENCES

[1] Ahuja, R.K., Magnanti, T.L. and Orlin, J.B., Network Flows, Prentice Hall, Englewood Cliffs, New Jersy, 1993.

[2] Hira, D.S. and Gupta, P.K., "Operation Research", S. Chand & Company Ltd.,1994.

[3] Ravindran, A., Phillips, Don T. and Solberg, James J., "Operations Research Principles And Practice", John Wiley & Sons, Second Edition.

[4] Sharma, K.D., "A New Approach for the simple transportation problem", unpublished M. Tech. Dissertation, Department of Industrial & Management Engineering, IIT Kanpur 208016, India, 1998.

[5] Prasad, Saumya, "Heuristic for obtaining good solution to uncapacitated transportation problem: An Empirical Investigation", unpublished M. Tech. Dissertation, Department of Industrial & Management Engineering, IIT Kanpur 208016, India, 2000.

[6] Sharma, R.R.K. and Sharma, K.D., " A new dual based procedure for the transportation problem", European Journal of Operation Research, 122(3), 2000, pp 37-48.

[7] Sharma, R.R.K. and Prasad, Saumya, " Obtaining a good primal solution to the uncapacitated transportation problem", European Journal of Operation Research, 144,2003,pp 550-554.

[8] Plotkin, S. and Tardos, E., "Improved dual network simplex", Proceedings of the First ACMN-SIAM Symposium on Discrete Algorithms, pp. 367376, 1990.

[9] Ali, A.I., Padman, R. and Thiagarajan, H., "Dual Algorithms for Pure Network Problems", Operations Research, V37, No.1, 159-171, 1989.

Appendix 1

dsc.c

```
/*DEMAND-SUPPLY MATRIX,RANDOM NUMBER GENERATOR*/
#include <stdio.h>
#include <stdlib.h>

 main()

{
     int i,t;
     int X=0,Y=0;  //X=no.of rows ie supply, Y =no of columns ie demand
     int Y1=0,X1=0;
     int value,new_val;
     int supply[201],demand[201];
     int sumX=0;
     int sumY=0;
     int diff=0;
     int l1=0,l2=0;
     int value1,new_val1;
     int seed;
 FILE *cost;
// FILE *cost_csv;
// FILE *ds;
 FILE *dmd_spl;
 FILE *dsc_csv;

 dmd_spl=fopen("dmd_spl.txt","w");
        //arrangements for inputting X,Y

// ds=fopen("ds.csv","w");
 dsc_csv=fopen("dsc_csv.csv","w");

 printf("Enter the value of no. of Supply:");
 scanf("%d",&X);
 printf("Enter the value of no  of Demand:");
 scanf("%d",&Y);

 printf("Enter the value of seed:");
 scanf("%d",&seed);
 srandom(seed);


 fprintf(dmd_spl,"%d\t%d\n",X,Y);
 fprintf(dsc_csv,"%d",X);
 fprintf(dsc_csv,",");
 fprintf(dsc_csv,"%d",Y);
 fprintf(dsc_csv,"\n");

 if ( X <= 2 && Y <= 2 )
   { if(X==0 || Y==0)
       printf("NOT POSSIBLE!!!, CHECK THE NO. OF SUPPLY & DEMAND.\n ");
     else if(X==1 && Y==1)
      {     new_val=0;
              do
```

```
            { value=rand();
              new_val = (abs)(value / 10000000);
            }
            while(new_val==0);
          supply[X]=new_val;
          demand[Y]=new_val;
       }
    else
       { for(t=1;t<=X;t++)
        { new_val=0;
            do
            { value=rand();
              new_val = (abs)(value / 10000000);
            }
            while(new_val==0);
          supply[t]=new_val;
            sumX=sumX+new_val;
          }
            if(Y==2)
            { demand[1]=(abs)(sumX/2);
              demand[2]=sumX-demand[1];
            }
            else if(Y==1)
            { demand[1]=sumX;
          }
        }//end of else on X=0 and Y=0
    }//end of if on x<=2 && y<=2


 else if((Y==1) && (X >=2) )
 {    for(i=1;i<=X;i++)        //generation of supply started
         { new_val=0;
         do
           { value=rand();
             new_val = (abs)(value / 10000000);
           }
           while(new_val==0);

         supply[i]=new_val;          //store supply in a array
           sumX=sumX+new_val;
         }
         demand[1]=sumX;

     }  //end of if(y<=2)

 else if((X==0) && (Y>=2))
     { printf("NOT POSSIBLE!!!, CHECK THE NO. OF SUPPLY & DEMAND.\n ");
     }

 else if((Y==0) && (X>=2))
     { printf("NOT POSSIBLE!!!, CHECK THE NO. OF SUPPLY & DEMAND.\n ");
     }

 else       //for X>=2 && Y>=2
   { Y1=(Y-2);
     while(sumX<(sumY+2))
     { sumX=0;sumY=0;
```

```
for(i=1;i<=X;i++)          //generation of supply started
    { new_val=0;
    do
      { value=rand();
        new_val = (abs)(value / 10000000);
      }
      while(new_val==0);

    supply[i]=new_val;          //store supply in a array
      sumX=sumX+new_val;
    }


        for (t=1;t<=Y1;t++)          //generation of demand started
      { new_val=0;
        do
        { value=rand();
          new_val = (abs)(value / 10000000);
        }
        while(new_val==0);

      demand[t]=new_val;
        sumY=sumY+new_val;
      }
}//end of while on the sum condition

diff=sumX-sumY;                 //Will reach here only sumX>sumY+2
l1 = (abs)(diff / 2);

if (diff==2*l1)
   l2=l1;
else
   l2=(l1+1);


demand[Y1+1]=l1;                //store the last 2 demands
demand[Y1+2]=l2;
}//end of else on X>=2 && Y>=2

for(t=1;t<=X;t++)
      {
              fprintf(dmd_spl,"%d\t",supply[t]);
      }

              fprintf(dmd_spl,"\n");
     for(t=1;t<=Y;t++)
      {
              fprintf(dmd_spl,"%d\t",demand[t]);
      }
 fprintf(dmd_spl,"\n");


     for(t=1;t<=X;t++)
     {
           fprintf(dsc_csv,"%d",supply[t]);
               if(t!=X) fprintf(dsc_csv,",");
     }
```

```
            fprintf(dsc_csv,"\n");
      for(t=1;t<=Y;t++)
      {
            fprintf(dsc_csv,"%d",demand[t]);
                if(t!=Y) fprintf(dsc_csv,",");
      }
  fprintf(dsc_csv,"\n\n");

//cost_csv.c

// int i,t;  //i=no. of rows,t=no. of columns.
// int value,new_val;

// FILE *cost;
 //FILE *pt;
// FILE *cost_csv;

// int X,Y;

 cost=fopen("cost.txt","w");
 //pt=fopen("dmd_spl.txt","r");
 //cost_csv=fopen("cost_csv.csv","w");

// fscanf(pt,"%d",&X);
 //fscanf(pt,"%d",&Y);

 for(i=1;i<=X;i++)
    { for (t=1;t<=Y;t++)
          { new_val1=0;
            do
            { value1=rand();
              new_val1 = (abs)(value1 / 10000000);
            }
            while(new_val1==0);
            fprintf(cost,"%d\t",new_val1);
          }
       fprintf(cost,"\n\n");

     }
 for(i=1;i<=X;i++)
    { for (t=1;t<=Y;t++)
          { new_val1=0;
            do
            { value1=rand();
              new_val1 = (abs)(value1 / 10000000);
            }
            while(new_val1==0);
            fprintf(dsc_csv,"%d",new_val1);
            if(t!=Y) fprintf(dsc_csv,",");
          }
       fprintf(dsc_csv,"\n");

     }


 }//End of main
```

Appendix 2

sharma input.c

```
/*CONVERSION OF COST AND DEM-SPL SO AS TO MAKE THEM SUITABLE FOR PRASAD CODE*/
#include <stdio.h>
#include <stdlib.h>
int Tot_Col=0,Tot_Row=0;
#define MAXVAL 201
main()
{

    FILE *pt;
    FILE *pt1;
    FILE *pt2;
    FILE *result_csv;

    int t,i,sum=0,temp=0;
    float sum_dem=0.0,sum_sup=0.0;

    int matrix[MAXVAL][MAXVAL];           //matrix=>cost matrix
    int supply[MAXVAL],demand[MAXVAL];
    int new_cost[MAXVAL][MAXVAL];
    float b[MAXVAL];
    float d[MAXVAL];

    pt2=fopen("dmd_spl.txt","r");
    pt=fopen("cost.txt","r");

    fscanf(pt2,"%d",&Tot_Row);
    fscanf(pt2,"%d",&Tot_Col);

   for(t=1;t<=Tot_Row;t++)
    {     for(i= 1;i<=Tot_Col;i++)
          { new_cost[t][i]=0;
          }
    }

    for(t=1;t<=Tot_Row;t++)
    {     for(i=1;i<=Tot_Col;i++)
          { fscanf(pt,"%d",&temp);          //This is to read the transport costs
             matrix[t][i]=temp;
          }
    }

    for(t=1;t<=Tot_Row;t++)
    {     fscanf(pt2,"%d",&supply[t]);     //This is to read the initial supplies
    }
    for(t=1;t<=Tot_Col;t++)
    {     fscanf(pt2,"%d",&demand[t]);     //This is to read the initial demands
    }

    close(pt2);

    close(pt);
```

```
    for(t=1;t<=Tot_Row;t++)
        { sum=sum+supply[t];
        }
    printf("sum=%d\n",sum);

    for(t=1;t<=(Tot_Row-1);t++)
        { b[t]=(float)supply[t]/(float)sum;
          sum_sup=sum_sup+b[t];
        }
    b[Tot_Row]=1.0-sum_sup;

    for(t=1;t<=(Tot_Col-1);t++)
        { d[t]=(float)demand[t]/(float)sum;
          sum_dem=sum_dem+d[t];
        }
    d[Tot_Col]=1.0-sum_dem;


    for(t=1;t<=Tot_Row;t++)
    {     for(i= 1;i<=Tot_Col;i++)
          {  new_cost[i][t]=matrix[t][i]*sum;
          }
    }


    pt1=fopen("dsc_sharma.txt","w");

    fprintf(pt1,"%d\t%d\n",Tot_Row,Tot_Col);

    for(t=1;t<=Tot_Row;t++)
        { fprintf(pt1,"%f\t",b[t]);
        }
    fprintf(pt1,"\n");

    for(t=1;t<=Tot_Col;t++)
        { fprintf(pt1,"%f\t",d[t]);
        }
    fprintf(pt1,"\n");

    for(t=1;t<=Tot_Row;t++)
    {     for(i= 1;i<=Tot_Col;i++)
          {         fprintf(pt1,"%d\t",new_cost[t][i]) ;
          }
          fprintf(pt1,"\n");
    }

    close(pt1);
}
```

```
    for(t=1;t<=Tot_Row;t++)
        { sum=sum+supply[t];
        }
    printf("sum=%d\n",sum);

    for(t=1;t<=(Tot_Row-1);t++)
        { b[t]=(float)supply[t]/(float)sum;
          sum_sup=sum_sup+b[t];
        }
    b[Tot_Row]=1 0-sum_sup;

    for(t=1;t<=(Tot_Col-1);t++)
        { d[t]=(float)demand[t]/(float)sum;
          sum_dem=sum_dem+d[t];
        }
    d[Tot_Col]=1.0-sum_dem;


    for(t=1;t<=Tot_Row;t++)
    {     for(i= 1;i<=Tot_Col;i++)
          {   new_cost[i][t]=matrix[t][i]*sum;
          }
    }


    pt1=fopen("dsc_sharma.txt","w");

    fprintf(pt1,"%d\t%d\n",Tot_Row,Tot_Col);

    for(t=1;t<=Tot_Row;t++)
        { fprintf(pt1,"%f\t",b[t]);
        }
    fprintf(pt1,"\n");

    for(t=1;t<=Tot_Col;t++)
        { fprintf(pt1,"%f\t",d[t]);
        }
    fprintf(pt1,"\n");

    for(t=1;t<=Tot_Row;t++)
    {     for(i= 1;i<=Tot_Col,i++)
          {         fprintf(pt1,"%d\t",new_cost[t][i])  ;
          }
          fprintf(pt1,"\n");
    }

    close(pt1);
}
```

Appendix 3

saumya.pas

```
PROGRAM Heuristic_for_STP(input,output);
{*****************************************************************************
**}
const
        I=250, { No. of plants }
        K=250; { No. of markets }
        Z=500; { No. of markets and palnts }
{****************************************************************************}

type    digits = SET of 1..255;

{****************************************************************************}

var
        c2,i1,k1,No_of_iteration,count : integer;
        ifmn : array [1..K,0..I] of integer;
      t · array[1..K,1..I] of real;
        b_boolean·array[1..I] of boolean;
        d_boolean:array[1..K] of boolean;
        b,original_b : array [1..I] of real;
        d,original_d · array [1..K] of real;
        vam_input,XIK,XIK_temp,original_BIK,BIK_plus_ZI : array [1..K,1..I] of
real;
        XIK_temp_not_zero,XIK_not_zero : array [1..K,1..I] of boolean;
        DK_and_BI,DK_and_BI_temp : array [1..Z,1..1] of real;
        DK_and_BI_indexed : array [1..Z,1 .4] of integer;
        row_column,yes_no_completed,numbered : integer;
        B3K,B1K, B2K : array [1..K] of real;
        z_for_increase : array [1..I] of integer;
        k_for_increase : array [1. k] of integer;
        p1,m1 : integer;
        value_of_zi : array [1..I] of real;
        obj_fn_value_using_v0_vk,v0 : real;
        value_of_vk : array [1..K] of real;
        Collection_of_zi_that_can_be_increased : array [0..I] of integer;
        min_in_col : array [0..K,1..I] of integer;
        single_zi_that_can_be_increased : integer;
        objective_fn_value, current_loss, cumulative_loss, extent_of_increase :
real;
        improvement_possible : Boolean;
        cross,minima: array [1..I,1..K] of boolean;
        optimal_sol : real;
        modified_vam_input, modified_BIK, modified_BIK_plus_ZI: array
 [1..I,1..K] of real;


 {****************************************************************************}

 procedure Initialization1;

 var
      i1,k1,index : integer;
```

```
Begin {initialization}
      for index·=1 to K do
      for i1:=0 to I do
      ifmn[index,i1]:=0;
      for index·=1 to I do
      value_of_zi[index].=0;
      for index:=1 to I do
      begin
            B1K[index]:=0;
            B2K[index]:=0;
     end;
     cumulative_loss:=0;
     current_loss:=0;
     extent_of_increase:=0;
End;{initialization}

{***********************************************************************}

procedure read_input;

var
       i1,k1 : integer;
       f1:text;

begin {read_input}
      assign(f1,'dsc_sharma.txt');
      reset(f1);
      read(f1,p1);
      readln(f1,m1);
      for i1:=1 to p1 do
      read(f1,b[i1]);
      readln(f1);
      for k1:=1 to m1 do
      read(f1,d[k1]);
      readln(f1);
      for k1:=1 to m1 do
      begin
           for i1:=1 to p1 do
           begin
                 read(f1,original_BIK[k1,i1]);
                 modified_BIK[i1,k1]·=original_BIK[k1,i1];
                 BIK_plus_zi[k1,i1]:=original_BIK[k1,i1];
           end;
      readln(f1);
      End;
      writeln;
      for i1:=1 to p1 do
         begin
              original_b[i1]:=b[i1];
             {* write(original_b[i1]·9:6,' ');*}
       end;
      {* writeln;*}
       for k1:=1 to m1 do
       begin
            original_d[k1]:=d[k1];
      {*    write(original_d[k1]:9:6,' ');*}
```

```
        end;
       {* writeln;*}

end; {read_input}
{**************************************************************************}


procedure Prepare_Set_ifmn_k;

var
      i1,k1 : integer;

begin {Prepare_set_ifmn_k}
      for k1:=1 to m1 do
      for i1:=1 to p1 do
      begin
            ifmn[k1,0]·=0;
            ifmn[k1,i1]:=0;
      end;
      for k1:=1 to m1 do
      begin
            B1K[k1]:=100000000;
            B2K[k1]:=9999999;
            ifmn[k1,0]:=0;
            for i1:=1 to p1 do
            begin
                  if (BIK_plus_zi[k1,i1]<B1K[k1]) then
                  begin
                        ifmn[k1,0]:=1;
                        ifmn[k1,1]:=i1;
                        B2K[k1]:=B1K[k1];
                        B1K[k1]:=BIK_plus_zi[k1,i1],
                  end
                  else if (BIK_plus_zi[k1,i1]=B1K[k1]) then
                  begin
                        ifmn[k1,0]:=ifmn[k1,0]+1;
                        ifmn[k1,ifmn[k1,0]]:=i1;
                  end
                  else if ((BIK_plus_zi[k1,i1]>B1K[k1]) and
(BIK_plus_zi[k1,i1]<B2K[k1])) then
                  B2K[k1]:=BIK_plus_zi[k1,i1];
            End;
      end;
end; {Prepare_set_ifmn_k}
{************************************************************************}

procedure Obtain_solution_with_all_zi_at_zero;

var
          i1, k1 : integer;

begin{Obtain_solution_with_all_zi_at_zero}
          for i1:=1 to p1 do
          value_of_zi[i1]:=0;
          objective_fn_value:=0;
          for k1:=1 to m1 do
          objective_fn_value:=objective_fn_value+B1K[k1]*d[k1];
```

```
end;{OBtain_solution_with_all_zi_at_zero}

{*************************************************************************}
procedure prepare_min_in_col_array;
var
        i1,k1,col_no,row_no : integer;

begin {prepare_min_in_col_array}
      for i1:=1 to p1 do
      begin
            min_in_col[0,i1]:=0;
            for k1:=1 to m1 do
            min_in_col[k1,i1]:=0;
      end;
      for k1:=1 to m1 do
      begin
            row_no:=ifmn[k1,0];
            for i1:=1 to row_no do
            begin
                  col_no:=ifmn[k1,i1];
                  min_in_col[0,col_no]:=min_in_col[0,col_no]+1;
                  min_in_col[min_in_col[0,col_no],col_no]:=k1;
            end;
      end;
end; {prepare_min_in_col_array}
{**********************************************************************}

Procedure Solution_improvement_using_sets;
var
      k1,i1 : integer;
      Benefit_1,Benefit_2,B1_check,B2_check : real;
      z_i,Union_of_ifmn_sets, group_set, K_set_for_group_set · digits;
      New_combination_found, New_member_k_is_added, Member_has_common_element :
Boolean;

begin {Solution_improvement_using_sets}
      Union_of_ifmn_sets:=[];
      for k1:=1 to m1 do
      for i1:=1 to ifmn[k1,0] do
      Union_of_ifmn_sets:=Union_of_ifmn_sets+[ifmn[k1,i1]];
      group_set:=[];
      for i1:=1 to ifmn[1,0] do
      group_set:=group_set+[ifmn[1,i1]];
      K_set_for_group_set:=[1];
      new_member_k_is_added:=true;
      while (new_member_k_is_added) do
      begin
          new_member_k_is_added:=false;
          for k1:=2 to m1 do
          if not (k1 in k_set_for_group_set) then
          begin
               Member_has_common_element·=false;
               for i1:=1 to ifmn[k1,0] do
               if (ifmn[k1,i1] in group_set) then
               Member_has_common_element:=true;
               if (Member_has_common_element) then
               begin
```

```
end;{OBtain_solution_with_all_zi_at_zero}

{**********************************************************************}
procedure prepare_min_in_col_array;
var
        i1,k1,col_no,row_no : integer;

begin {prepare_min_in_col_array}
      for i1:=1 to p1 do
      begin
            min_in_col[0,i1]:=0;
            for k1:=1 to m1 do
            min_in_col[k1,i1]:=0;
      end;
      for k1:=1 to m1 do
      begin
            row_no:=ifmn[k1,0];
            for i1:=1 to row_no do
            begin
                  col_no:=ifmn[k1,i1];
                  min_in_col[0,col_no]:=min_in_col[0,col_no]+1;
                  min_in_col[min_in_col[0,col_no],col_no]:=k1;
            end;
      end;
end; {prepare_min_in_col_array}
{**********************************************************************}

Procedure Solution_improvement_using_sets;
var
      k1,i1 : integer;
      Benefit_1,Benefit_2,B1_check,B2_check : real;
      z_i,Union_of_ifmn_sets, group_set, K_set_for_group_set : digits;
      New_combination_found, New_member_k_is_added, Member_has_common_element :
Boolean;

begin {Solution_improvement_using_sets}
      Union_of_ifmn_sets:=[];
      for k1:=1 to m1 do
      for i1:=1 to ifmn[k1,0] do
      Union_of_ifmn_sets:=Union_of_ifmn_sets+[ifmn[k1,i1]];
      group_set:=[];
      for i1:=1 to ifmn[1,0] do
      group_set:=group_set+[ifmn[1,i1]];
      K_set_for_group_set:=[1];
      new_member_k_is_added:=true;
      while (new_member_k_is_added) do
      begin
           new_member_k_is_added:=false;
           for k1:=2 to m1 do
           if not (k1 in k_set_for_group_set) then
           begin
                Member_has_common_element:=false;
                for i1:=1 to ifmn[k1,0] do
                if (ifmn[k1,i1] in group_set) then
                Member_has_common_element:=true;
                if (Member_has_common_element) then
                begin
```

```
                    for i1:=1 to ifmn[k1,0] do
                    group_set:=group_set+[ifmn[k1,i1]];
                    k_set_for_group_set:=k_set_for_group_set+[k1];
                    new_member_k_is_added:=true;
               end;
          end;
     end;
               Benefit_1:=0.0;
               Benefit_2:=0.0;
     z_i:=[];
     for i1:=1 to p1 do
     z_i:=z_i+[i1];
     if (group_set=z_i) then
     begin
         { writeln('The solution can not be improved further using set
heuristic');}
         {  writeln;}
          improvement_possible:=false;
     end
     else begin
             for k1:=1 to m1 do
             if (k1 in k_set_for_group_set) then
             Benefit_1:=Benefit_1+d[k1]
             else Benefit_2:=Benefit_2+d[k1];
             for i1:=1 to p1 do
             if (i1 in group_set) then
             Benefit_1:=Benefit_1-b[i1]
             else if (i1 in (Union_of_ifmn_sets-group_set)) then
             Benefit_2:=Benefit_2-b[i1];
         end;
             new_combination_found:=false;
             for i1:=1 to I do
             z_for_increase[i1]:=0;
             for k1:=1 to K do
             k_for_increase[k1]:=0;
             B1_check:=0;
             B2_check:=0;
             B1_check:=round(100000*Benefit_1);
             B2_check:=round(100000*Benefit_2);
             if ((Benefit_1>0.0) and (B1_check>0.0)) then
             begin
                 for i1:=1 to p1 do
                 if (i1 in group_set) then
                 z_for_increase[i1]:=1;
                 for k1:=1 to m1 do
                 if (k1 in k_set_for_group_set) then
                 k_for_increase[k1]:=1;
                 new_combination_found:=true;
             end else if ((Benefit_2>0.0) and (B2_check>0.0)) then
             begin
                 for i1:=1 to p1 do
                 if (i1 in (union_of_ifmn_sets-group_set)) then
                 z_for_increase[i1]:=1;
                 for k1:=1 to m1 do
                 if not (k1 in k_set_for_group_set) then
                 k_for_increase[k1]:=1;
                 new_combination_found:=true;
```

```
                    end;
                    if not (new_combination_found) then
                    begin
                          improvement_possible:=false;
                        {   writeln('This Heuristic Using sets terminates here');}
                    end else
                    begin
                         improvement_possible:=true;
                         for i1:=0 to I do
                         collection_of_zi_that_can_be_increased[i1]:=0;
                         for i1:=1 to p1 do
                         if (z_for_increase[i1]=1) then
                         begin

collection_of_zi_that_can_be_increased[0]:=collection_of_zi_that_can_be_increase
d[0]+1;

collection_of_zi_that_can_be_increased[collection_of_zi_that_can_be_increased[0]
]:=i1;
                         end;
                    end;
end; {Solution_improvement_using_sets}
{***********************************************************************}

procedure Try_other_combinations;
var
       temp,k1,k2,k3,k4,i1,i2,i3,i4 : integer;
       lb_check,Tb_check,local_benefit,total_benefit : real;
       sorted_ifmn_k : array [1..k] of integer;
       combination_found : Boolean;

begin {Try_other_combinations}
       lb_check:=0.0;
       Tb_check:=0.0;
       for k1:=1 to m1 do
       sorted_ifmn_k[k1]:=k1;
       for k1:=1 to (m1-1) do
       for k2:=(k1+1) to m1 do
       if (ifmn[sorted_ifmn_k[k1],0]<ifmn[sorted_ifmn_k[k2],0]) then
       begin
             temp:=sorted_ifmn_k[k1];
             sorted_ifmn_k[k1]:=sorted_ifmn_k[k2];
             sorted_ifmn_k[k2]:=temp;
       end;
       k1:=0;
       combination_found:=false;
       while ((k1<m1) and not(combination_found)) do
       begin
            k1:=k1+1;
            for i1:=1 to I do
            z_for_increase[i1]:=0;
            for k3:=1 to K do
            k_for_increase[k3]:=0;
            k2:=sorted_ifmn_k[k1];
            if (ifmn[k2,0]>=2) then
            begin
                  total_benefit:=0;
```

```
                for i1:=1 to ifmn[k2,0] do
                begin
                     total_benefit:=total_benefit+(-1)*b[ifmn[k2,i1]];
                     z_for_increase[ifmn[k2,i1]]:=1;
                end;
                total_benefit:=total_benefit+d[k2];
                k_for_increase[k2]:=1;
                for k3:=(k1+1) to m1 do
                begin
                     k4:=sorted_ifmn_k[k3];
                     local_benefit:=d[k4];
                     for i2:=1 to ifmn[k4,0] do
                     begin
                          i3:=ifmn[k4,i2];
                          if (z_for_increase[i3]=0) then
                          local_benefit:=local_benefit-b[i3];
                     end;
                     lb_check:=round(100000*local_benefit);
                     if ((local_benefit>0.0) and (lb_check>0.0)) then
                     begin
                          for i2:=1 to ifmn[k4,0] do
                          begin
                               i3:=ifmn[k4,i2];
                               z_for_increase[i3]:=1;
                          end;
                          k_for_increase[k4]:=1;
                          total_benefit:=total_benefit+local_benefit;
                     end;
                end;
                Tb_check:=round(100000*Total_benefit);
                if ((total_benefit>0.0) and (Tb_check>0.0)) then
                combination_found:=true;
           end;
      end;
      if (not (combination_found)) then
                 begin
                 Solution_improvement_using_sets;
                { writeln;}
                 end
      else begin
                 improvement_possible:=true;
                 for i1:=0 to I do
                 collection_of_zi_that_can_be_increased[i1]:=0;
                 for i1:=1 to p1 do
                 if (z_for_increase[i1]=1) then
                 begin

collection_of_zi_that_can_be_increased[0]:=collection_of_zi_that_can_be_increase
d[0]+1;

collection_of_zi_that_can_be_increased[collection_of_zi_that_can_be_increased[0]
]:=i1;
                 end;
           end;

end; {Try_other_combinations}
{*********************************************************************}
```

```
procedure Dual_solution_can_be_improved;

var
        Benefit_for_single_zi : array [1..I] of real;
        index_no, index_i : integer;
        k_value,no_in_col_with_min_value : integer;
        B_check : real;

begin {Dual_solution_can_be_improved}
        B_check:=0.0;
        improvement_possible:=false;
        for index_i:=1 to p1 do
        Benefit_for_single_zi[index_i]:=-1*b[index_i];
        index_i:=0;
        repeat
              index_i:=index_i+1;
              no_in_col_with_min_value:=min_in_col[0,index_i];
              for index_no:=1 to no_in_col_with_min_value do
              begin
                   k_value:=min_in_col[index_no,index_i];
                   if (ifmn[k_value,0]=1) then

Benefit_for_single_zi[index_i]:=Benefit_for_single_zi[index_i]+d[k_value];
              end;
       B_check:=round(100000*benefit_for_single_zi[index_i]);
       until (((benefit_for_single_zi[index_i]>0) and (B_check>0.0)) or
(index_i=p1));
       B_check:=0.0;
       B_check:=round(100000*benefit_for_single_zi[index_i]);
       if ((Benefit_for_single_zi[index_i]>0) and (B_check>0.0)) then
       begin
            improvement_possible:=true;
            collection_of_zi_that_can_be_increased[0]:=1;
            collection_of_zi_that_can_be_increased[1]:=index_i;
      end
      else Try_other_combinations;

end;{Dual_solution_can_be_improved}
{*******************************************************************}

procedure Determine_extent_of_increase;

var
     zr,i2,k1,i1,temp : integer;
     zivr : array [1..k] of real;
     ks_for_positive_zivr : array [0..k] of integer;
     B_check,Benefit : real;

begin {Determine_extent_of_increase}
     B_check:=0.0;
     extent_of_increase:=0;
     ks_for_positive_zivr[0]:=0;
     for k1:=1 to m1 do
     begin
     ks_for_positive_zivr[k1]:=0;
     zivr[k1]:=0;
```

```
    end;
    if (collection_of_zi_that_can_be_increased[0]=1) then
    begin
          zr:=collection_of_zi_that_can_be_increased[1];
          for k1:=1 to m1 do
          if ((ifmn[k1,0]=1) and (ifmn[k1,1]=zr)) then
          begin
                zivr[k1]:=B2k[k1]-B1k[k1];
                ks_for_positive_zivr[0]:=ks_for_positive_zivr[0]+1;
                ks_for_positive_zivr[ks_for_positive_zivr[0]]:=k1;
          end;
    end else
    begin
          for k1:=1 to k do
          B3K[k1]:=0;
          for k1:=1 to m1 do
          begin
               B3K[k1]:=1000000000;
               for i1:=1 to p1 do
               if (z_for_increase[i1]=0) then
               begin
                    if (BIK_plus_zi[k1,i1]<B3K[k1]) then
                    B3K[k1]:=BIK_plus_zi[k1,i1];
               end;
               if (k_for_increase[k1]=1) then
               begin
                    zivr[k1]:=B3K[k1]-B1K[k1];
                    ks_for_positive_zivr[0]:=ks_for_positive_zivr[0]+1;
                    ks_for_positive_zivr[ks_for_positive_zivr[0]]:=k1;
               end;
          end;
     end;
          for i1:=1 to (ks_for_positive_zivr[0]-1) do
          for k1:=(i1+1) to (ks_for_positive_zivr[0]) do
          begin
               if
(zivr[ks_for_positive_zivr[i1]]>zivr[ks_for_positive_zivr[k1]])
               then begin
                       temp:=ks_for_positive_zivr[i1];
                       ks_for_positive_zivr[i1]:=ks_for_positive_zivr[k1];
                       ks_for_positive_zivr[k1]:=temp;
                    end;
          end;
          temp:=collection_of_zi_that_can_be_increased[0];
          Benefit:=0;
          for i1:=1 to temp do
          begin
          i2:= collection_of_zi_that_can_be_increased[i1];
          Benefit:=Benefit-b[i2];
          end;
          for k1:=1 to ks_for_positive_zivr[0] do
          Benefit:=Benefit+d[ks_for_positive_zivr[k1]];
          k1:=0;
          B_check:=round(100000*Benefit);
          while ((Benefit>=0) and (B_check>0.0) and
 (k1<=ks_for_positive_zivr[0])) do
          begin
```

```
                k1:=k1+1;
                Benefit:=Benefit-d[ks_for_positive_zivr[k1]];
          end;
          extent_of_increase:=zivr[ks_for_positive_zivr[k1]];
          for i1:=1 to temp do
          begin
          i2:=collection_of_zi_that_can_be_increased[i1];
          value_of_zi[i2]:=value_of_zi[i2]+extent_of_increase;
          end;
end;{Determine_extent_of_increase}
{***********************************************************************}

procedure improve_the_solution;
var
      i2,k2 : integer;

begin {improve_the_solution}
      current_loss:=0;
      for i2:=1 to collection_of_zi_that_can_be_increased[0] do
      begin
           current_loss:=current_loss-
b[collection_of_zi_that_can_be_increased[i2]]*extent_of_increase;
           for k2:=1 to m1 do

BIK_plus_zi[k2,collection_of_zi_that_can_be_increased[i2]]:=BIK_plus_zi[k2,colle
ction_of_zi_that_can_be_increased[i2]]+extent_of_increase;
      end;
end; {improve_the_solution}
{*****************************************************************}

procedure Compute_the_improved_solution;
var
     k2 : integer;

begin {Compute_the_improved_solution}
      objective_fn_value:=0;
      for k2:=1 to m1 do
      objective_fn_value:=objective_fn_value+B1K[k2]*d[k2];
      cumulative_loss:=cumulative_loss+current_loss;
      Objective_fn_value:=objective_fn_value+cumulative_loss;
end; {Compute_the_improved_solution}
{******************************************************************}
procedure Dual_var_calculation;
var
     i1,k1 : integer;
     larg,obj  : real;

begin {Dual_var_calculation}
      for k1:=1 to m1 do
      value_of_vk[k1]:=0;
      v0:=0;
      larg:=0;
      for k1:=1 to m1 do
      if (B1K[k1]>larg) then larg:=B1K[k1];
      v0:=larg;
      obj:=0;
      for k1:=1 to m1 do
```

```
      begin
           value_of_vk[k1]:=v0-B1K[k1];
           obj:=obj-value_of_vk[k1]*d[k1];
      end;
      obj_fn_value_using_v0_vk:=v0+obj+cumulative_loss;
end; {Dual_var_calculation}
{*****************************************************************}
procedure temp_for_saumya;
var
    i1,k1:integer;
    fn:text;
begin
    assign(fn,'slack.txt');
    rewrite(fn);
  for k1:=1 to m1 do
     for i1:=1 to p1 do
       begin
        modified_BIK_plus_zi[i1,k1]:=BIK_plus_zi[k1,i1];
       end;

  for k1:=1 to k1 do
     for i1:=1 to p1 do
       begin
        vam_input[k1,i1]:= original_BIK[k1,i1]-
v0+value_of_zi[i1]+value_of_vk[k1];
        modified_vam_input[i1,k1]:= vam_input[k1,i1];
       end;

   for k1:=1 to k1 do
     begin
     for i1:=1 to p1 do
       begin
        write(fn,modified_vam_input[k1,i1]:6:2,' ');
       end;
     writeln(fn);
     end;
     close(fn);
end;


{*******************************************************************}

procedure primal_sol_calculation;
var

c2,i1,k1,i,j,found,count,equation_no,temp,sum,total_no_of_variables_to_be_comput
ed: integer;
      n,loc,l : 1..100;
      sum1,temp1,min,optimal_sol,min1,temp_sol,temp_optimal_sol:real;
      numbering_completed,to_be_computed:boolean;

begin {making true of the corresponding primal values}
{writeln('**********************************************');}

  for k1:=1 to m1 do
     for i1:=1 to p1 do
       begin
```

```
            XIK_not_zero[k1,i1]:=false;
        end;

   optimal_sol:=0;
   total_no_of_variables_to_be_computed:=0;

 for k1:=1 to m1 do
   begin
     for i1:=1 to p1 do
     begin
       if (BIK_plus_zi[k1,i1]=B1K[k1]) then
         begin
           XIK_not_zero[k1,i1]:=true;

total_no_of_variables_to_be_computed:=total_no_of_variables_to_be_computed+1;
         end;
     end;
   end;

{**********************************************************}

{preparation of the array of row_column and DK_and_BI values}

i:=0; j:=0;
row_column:=0;  {0 # row, 1 # column}
for k1:=1 to m1 do
    begin
      DK_and_BI[k1,1]:=d[k1];
      DK_and_BI_indexed[k1,1]:=row_column;
    end;
k1:=0; i1:=0;
row_column :=1;
for i1:=1 to p1 do
    begin
      DK_and_BI[m1+i1,1]:=b[i1];
      DK_and_BI_indexed[m1+i1,1]:=row_column;
    end;
k1:=0; i1:=0;
numbered :=0; {1 # numbered, 0 # not_numbered}
for i:=1 to m1+p1 do
     begin
       DK_and_BI_indexed[i,4]:=numbered;
       DK_and_BI_indexed[i,2]:=0;
     end;

No_of_iteration:=0;
sum:=total_no_of_variables_to_be_computed;


while(sum<>0) do
begin
 i:=0;
 min:=1;

  for l:=1 to m1+p1 do
    begin
      DK_and_BI_indexed[l,2]:=0;
```

```
      DK_and_BI_indexed[l,3]:=0;
      if (DK_and_BI_indexed[l,4]=1) then
            DK_and_BI_indexed[l,4]:=0;
    end;

 for l:=1 to m1+p1 do
     begin
       for loc:=1 to m1+p1 do
         begin
           if ((DK_and_BI_indexed[loc,4]=0)and(DK_and_BI[loc,1]<min)) then
           begin
           i:=loc;
           min:=DK_and_BI[i,1];
           found:=1;
           end;
         end;
       DK_and_BI_indexed[i,4]:=1;
       DK_and_BI_indexed[i,2]:=1;
       DK_and_BI_indexed[l,3]:=i;
       min:=1;
     end;
 i:=0;
 to_be_computed:=true;
while(to_be_computed=true) do
  begin
     i:=i+1;
     equation_no:=DK_and_BI_indexed[i,3];
     if((DK_and_BI_indexed[equation_no,1]=0) and
(DK_and_BI_indexed[equation_no,4]=1) and (d[equation_no]>0))then
        begin{row no remains the same}
         count:=0;
           for j:=1 to p1 do
              begin
                if (XIK_not_zero[equation_no,j]=true) {true i.e. that have
values}then
                  begin
                    count:= count+1;
                    found:=j;
                  end;
              end;
           if(count=1) then {assign value}
              begin
                XIK[equation_no,found]:=DK_and_BI[equation_no,1]; {assign dk}

optimal_sol:=XIK[equation_no,found]*original_BIK[equation_no,found];
                DK_and_BI[equation_no,1]:=0;
                DK_and_BI[m1+found,1]:=DK_and_BI[m1+found,1]-d[equation_no];
                DK_and_BI_indexed[equation_no,4]:=2;
                XIK_not_zero[equation_no,found]:=false; {replace true by False}
                b[found]:= b[found] - d[equation_no];
                d[equation_no]:=0;
                to_be_computed:=false;
                sum:=sum-1;
              end
           else if (count>1) then
              begin
                for j:=1 to p1 do
```

```
                    begin
                      if ((XIK_not_zero[equation_no,j]=true)and(j<>found)) then
                        begin
                         XIK[equation_no,j]:=0; {assign dk}
                         XIK_not_zero[equation_no,j]:=false; {replace true by
False}

                         sum:=sum-1;
                         end;
                    end;
                XIK[equation_no,found]:=DK_and_BI[equation_no,1]; {assign dk}

optimal_sol:=XIK[equation_no,found]*original_BIK[equation_no,found];
                DK_and_BI[equation_no,1]:=0;
                XIK_not_zero[equation_no,found]:=false;
                DK_and_BI[m1+found,1]:=DK_and_BI[m1+found,1]-d[equation_no];
                DK_and_BI_indexed[equation_no,4]:=2;
                b[found]:= b[found] - d[equation_no];
                d[equation_no]:=0;
                to_be_computed:=false;
                sum:=sum-1;
                end;
         end;

  if((DK_and_BI_indexed[equation_no,1]=1) and
(DK_and_BI_indexed[equation_no,4]=1) and (b[equation_no-m1]>0))then
        begin
         equation_no:=equation_no - m1;
         count:=0;
           for j:=1 to m1 do
              begin
                if (XIK_not_zero[j,equation_no]=true) {true i.e. have
values}then
                  begin
                    count:=count+1;
                    found:=j;
                  end;
              end;
           if (count=1) then {assign values}
             begin
               XIK[found,equation_no]:=DK_and_BI[equation_no+m1,1];

optimal_sol:=XIK[found,equation_no]*original_BIK[found,equation_no];
               DK_and_BI[equation_no+m1,1]:=0;
               DK_and_BI[found,1]:=DK_and_BI[found,1]-b[equation_no];
               DK_and_BI_indexed[equation_no+m1,4] :=2;
               XIK_not_zero[found,equation_no] := false;
               d[found]:=d[found]- b[equation_no];
               b[equation_no]:=0;
               to_be_computed:=false;
               sum:=sum-1;
             end
           else if (count>1) then
             begin
               for j:=1 to m1 do
                 begin
                   if ((XIK_not_zero[j,equation_no]=true)and(j<>found)) then
                     begin
```

```
                            XIK[j,equation_no]:=0;
                            XIK_not_zero[j,equation_no] := false;
                            sum:=sum-1;
                          end;
                      end;
                 XIK[found,equation_no]:=DK_and_BI[equation_no+m1,1];

optimal_sol:=XIK[found,equation_no]*original_BIK[found,equation_no];
                 DK_and_BI[equation_no+m1,1]:=0;
                 XIK_not_zero[found,equation_no] := false;
                 DK_and_BI[found,1]:=DK_and_BI[found,1]-b[equation_no];
                 DK_and_BI_indexed[equation_no+m1,4] :=2;
                 d[found]:=d[found]- b[equation_no];
                 b[equation_no]:=0;
                 to_be_computed:=false;
                 sum:=sum-1;
                 end;
           end;
  end;
No_of_iteration:=No_of_iteration+1;
end;
{********************************************************}
temp_optimal_sol:=optimal_sol;
for i1:=1 to p1 do
  for k1:=1 to m1 do
     begin
      XIK_temp[k1,i1]:=XIK[k1,i1];
      XIK_temp_not_zero[k1,i1]:=XIK_not_zero[k1,i1];
     end;
for i:=1 to m1+p1 do
   DK_and_BI_temp[i,1]:=DK_and_BI[i,1];
{********************************************************}
for l:=1 to m1+p1 do
begin
  DK_and_BI_indexed[l,4]:=0;
end;
sum1:=0;
for i:=1 to m1+p1 do
    begin
      if(DK_and_BI[i,1]>0)then
      sum1:=sum1+1;
    end;

while(sum1 >= 2) do
begin
  for l:=1 to m1+p1 do
    begin
     DK_and_BI_indexed[l,2]:=0;
     DK_and_BI_indexed[l,3]:=0;
     if (DK_and_BI_indexed[l,4]=1) then
             DK_and_BI_indexed[l,4]:=0;
     end;
min:=1;
for l:=1 to m1+p1 do
     begin
       for loc:=1 to m1+p1 do
         begin
```

```
            if
((DK_and_BI_indexed[loc,4]=0)and(DK_and_BI[loc,1]<min)and(DK_and_BI[loc,1]>0))
then
            begin
            i:=loc;
            min:=DK_and_BI[i,1];
            found:=1;
            end;
          end;
        DK_and_BI_indexed[i,4]:=1;
        DK_and_BI_indexed[i,2]:=1;
        DK_and_BI_indexed[1,3]:=i;
        min:=1;
      end;

i:=0;to_be_computed:=true;
while(to_be_computed=true) do
begin
  i:=i+1;
  j:=i;
   begin
    if (DK_and_BI_indexed[i,3]<=m1) then
      begin
      i:=DK_and_BI_indexed[i,3];
      min1:=999999;
       for loc:=1 to p1 do
         begin
           if (original_BIK[i,loc]<min1)and(DK_and_BI[loc+m1,1]>0) then
           begin
           l:=loc;
           min1:=original_BIK[i,l];
           end;
         end;
         XIK[i,l]:=XIK[i,l]+DK_and_BI[i,1];
         temp1:=DK_and_BI[i,1] * original_BIK[i,l];
         optimal_sol:=optimal_sol + temp1;
         min1:=99999;
         DK_and_BI[l+m1,1]:=DK_and_BI[l+m1,1]-DK_and_BI[i,1];
         DK_and_BI[i,1]:=0;
         b[l]:=b[l]-d[i];
         d[i]:=0;
         to_be_computed:=false;
      end;
      i:=j;
    if (DK_and_BI_indexed[i,3]>m1) then
       begin
       i:=DK_and_BI_indexed[i,3]-m1;
       min1:=999999;
       for loc:=1 to m1 do
         begin
           if ((original_BIK[loc,i]<min1)and(DK_and_BI[loc,1]>0)) then
           begin
           l:=loc;
           min1:=original_BIK[l,i];
           end;
         end;
         XIK[l,i]:=XIK[l,i]+DK_and_BI[i+m1,1];
```

```
          temp1:=DK_and_BI[i+m1,1] * original_BIK[1,i];
          optimal_sol:=optimal_sol + temp1;
          min1:=99999;
          DK_and_BI[1,1]:=DK_and_BI[1,1]-DK_and_BI[i+m1,1];
          DK_and_BI[i+m1,1]:=0;
          d[1]:=d[1]-b[i];
          b[i]:=0;
          to_be_computed:=false;
       end;
sum1:=0;
for i:=1 to m1+p1 do
    begin
      if(DK_and_BI[i,1]>0)then
      sum1:=sum1+1;
    end;
  end;
end;
end;
optimal_sol:=0;
for i:=1 to m1 do
  for j:=1 to p1 do
     begin
      optimal_sol :=optimal_sol+XIK[i,j]*original_BIK[i,j];
     end;
write(optimal_sol:5:2,'optimal cost mids');


{*********************************************************}

(*optimal_sol:=temp_optimal_sol;

for i1:=1 to p1 do
  for k1:=1 to m1 do
      XIK[k1,i1]:=XIK_temp[k1,i1];
for i:=1 to m1+p1 do
   DK_and_BI[i,1]:=DK_and_BI_temp[i,1];

for l:=1 to m1+p1 do
begin
  DK_and_BI_indexed[l,4]:=0;
end;
sum:=0;
for i:=1 to m1+p1 do
    begin
      if(DK_and_BI[i,1]>0)then
      sum1:=sum1+1;
    end;
while(sum1 >= 2) do
begin
    for l:=1 to m1+p1 do
      begin
          DK_and_BI_indexed[l,2]:=0;
          DK_and_BI_indexed[l,3]:=0;
          if (DK_and_BI_indexed[l,4]=1) then
                DK_and_BI_indexed[l,4]:=0;
       end;
  min:=1;
```

```
for l:=1 to m1+p1 do
   begin
     for loc:=1 to m1+p1 do
       begin
         if
((DK_and_BI_indexed[loc,4]=0)and(DK_and_BI[loc,1]<min)and(DK_and_BI[loc,1]>0))
then
         begin
         i:=loc;
         min:=DK_and_BI[i,1];
         found:=1;
         end;
       end;
     DK_and_BI_indexed[i,4]:=1;
     DK_and_BI_indexed[i,2]:=1;
     DK_and_BI_indexed[l,3]:=i;
     min:=1;
   end;

i:=0;
to_be_computed:=true;
while(to_be_computed=true) do
begin
  i:=i+1;
  j:=i;
  begin
    if (DK_and_BI_indexed[i,3]<=m1) then
       begin
       i:=DK_and_BI_indexed[i,3];
       min1:=999999;
       for loc:=1 to p1 do
         begin
           if (BIK_plus_zi[i,loc]<min1)and(DK_and_BI[loc+m1,1]>0) then
           begin
           l:=loc;
           min1:=BIK_plus_zi[i,l];
           end;
         end;
       XIK[i,l]:=XIK[i,l]+DK_and_BI[i,1];
       temp1:=DK_and_BI[i,1] * original_BIK[i,l];
       optimal_sol:=optimal_sol + temp1;
       min1:=99999;
       DK_and_BI[l+m1,1]:=DK_and_BI[l+m1,1]-DK_and_BI[i,1];
       DK_and_BI[i,1]:=0;
       b[l]:=b[l]-d[i];
       d[i]:=0;
       to_be_computed:=false;
       end;
    i:=j;
    if (DK_and_BI_indexed[i,3]>m1) then
       begin
       i:=DK_and_BI_indexed[i,3]-m1;
       min1:=999999;
       for loc:=1 to m1 do
         begin
           if ((BIK_plus_zi[loc,i]<min1)and(DK_and_BI[loc,1]>0)) then
           begin
```

```
            l:=loc;
            min1:=BIK_plus_zi[l,i];
            end;
          end;
        XIK[l,i]:=XIK[l,i]+DK_and_BI[i+m1,1];
        temp1:=DK_and_BI[i+m1,1] * original_BIK[l,i];
        optimal_sol:=optimal_sol + temp1;
        min1:=99999;
        DK_and_BI[l,1]:=DK_and_BI[l,1]-DK_and_BI[i+m1,1];
        DK_and_BI[i+m1,1]:=0;
        d[l]:=d[l]-b[i];
        b[i]:=0;
        to_be_computed:=false;
      end;
  sum1:=0;
  for i:=1 to m1+p1 do
    begin
      if(DK_and_BI[i,1]>0)then
      sum1:=sum1+1;
    end;
  end;
end;
end;

optimal_sol:=0;
for i:=1 to m1 do
  for j:=1 to p1 do
     begin
      optimal_sol :=optimal_sol+XIK[i,j]*original_BIK[i,j];
     end;

write(' ', optimal_sol:5:2);*)

end;

{*****************************************************************************
*******}


proceDure vam;

var
      i1,k1,i,j,l,loc,count : integer;
      min,max,temp:real;
      difference : array [1..Z] of real;
      all_is_crossed:boolean;
      small : array [1..Z,1..2] of real;
      f2 : text;
begin
   assign(f2,'input_optimal_vam.txt');
   rewrite(f2);

  for i1:=1 to p1 do
    begin
       for k1:=1 to m1 do
         begin
                XIK[i1,k1]:=0;
```

```
                    XIK_not_zero[i1,k1]:=false;
                    cross[i1,k1]:=false;
                t[i1,k1]:=0;
             end;
        end;

    for i1:=1 to p1 do
        begin
           for k1:=1 to m1 do
             begin
                    t[i1,k1]:=modified_BIK[i1,k1];
             end;
        end;
    {*
    writeln;
    for i1:=1 to p1 do
        begin
           for k1:=1 to m1 do
             begin
               write(t[i1,k1]:9:6,' ');
           end;
           writeln;
        end;
    writeln;
    *}
    {*writeln;*}
    for i1:=1 to p1 do
      begin
        b[i1]:=original_b[i1];
        {*write(b[i1]:9:6,' ');*}
      end;
     {* writeln;*}
    for k1:=1 to m1 do
      begin
        d[k1]:=original_d[k1];
        {*write(d[k1]:9:6,' ');*}
      end;
      {*writeln;*}
    optimal_sol:=0;

    all_is_crossed:=false;
    while(all_is_crossed=false) do
    begin
    {*********making false each time for finding the minima********}
    for i1:=1 to p1 do
        begin
           for k1:=1 to m1 do
             begin
               minima[i1,k1]:=false;
             end;
        end;
    {*****finding the first and second minimum in a row************}
    min:=9999999;
    temp:=-1;
    for i1:=1 to p1 do
       begin
         for l:=1 to 2 do
```

```
      begin
        for loc:=1 to m1 do
          begin
            if
((t[i1,loc]<min)and(minima[i1,loc]=false)and(cross[i1,loc]=false)and((temp=-
1)or(t[i1,loc]<>temp))) then

            begin
            i:=loc;
            min:=t[i1,loc];
            end;
          end;
          minima[i1,i]:=true;
          small[i1,l]:=min;
          temp:=small[i1,l];
          min:=9999999;
      end;
  end;

{*****finding the first and second minimum in a column*******}
for i:=1 to p1 do
  begin
  for j:=1 to m1 do
      begin
      minima[i,j]:=false;
      end;
  end;

min:=9999999;
temp:=-1;
for k1:=1 to m1 do
  begin
    for l:=1 to 2 do
      begin
        for loc:=1 to p1 do
          begin
            if
((t[loc,k1]<min)and(minima[loc,k1]=false)and(cross[loc,k1]=false)and((temp=-1)
or (t[loc,k1]<>temp))) then
            begin
            j:=loc;
            min:=t[loc,k1];
            end;
          end;
          minima[j,k1]:=true;
          small[p1+k1,l]:=min;
          temp:=small[p1+k1,l];
          min:=9999999;
      end;
  end;
{****************compute the difference*********************}

for i:=1 to p1+m1 do
    begin
      difference[i]:=small[i,2]-small[i,1];
    end;
```

```
{**********finding the maximum of difference*****************}
max:=0;
for i:=1 to p1+m1 do
    begin
      if (difference[i]>max) then
         begin
            max:=difference[i];
            loc:=i;
         end;
    end;

{****if loc is row then finding the cell with minimum cost ***}

if (loc<=p1) then
begin
   i1:=loc;
   min:=9999999;
   for i:=1 to m1 do
     begin
      if ((t[loc,i]<min)and(cross[loc,i]=false)) then
         begin
           l:=i;
           min:=t[loc,l];
         end;
     end;
   k1:=l;
end;

{****if loc is  column then finding the cell with minimum cost ***}

if (loc>p1) then
begin
   min:=9999999;
   k1:=loc-p1;
   for j:=1 to p1 do
     begin
       if ((t[j,k1]<min)and(cross[j,k1]=false)) then
          begin
            l:=j;
            min:=t[l,k1];
          end;
     end;
   i1:=l;
end;

      if (b[i1]=d[k1]) then
        begin
          XIK[i1,k1]:=d[k1];
          for i:=1 to m1 do
          cross[i1,i]:=true;
          for j:=1 to p1 do
          cross[j,k1]:=true;
          b[i1]:=0;
          d[k1]:=0;
        end;
      if (d[k1]<b[i1]) then
        begin
```

```
          XIK[i1,k1]:=d[k1];
          for j:=1 to p1 do
          cross[j,k1]:=true;
          b[i1]:=b[i1]-d[k1];
          d[k1]:=0;
        end;
      if (d[k1]>b[i1]) then
        begin
          XIK[i1,k1]:=b[i1];
          for i:=1 to m1 do
          cross[i1,i]:=true;
          d[k1]:=d[k1]-b[i1];
          b[i1]:=0;
        end;
count:=0;
for i:=1 to p1 do
    begin
      for j:=1 to m1 do
          begin
            if (cross[i,j]=true) then
               begin
               count:=count+1;
               end;
          end;
    end;
if (count=m1*p1) then all_is_crossed:=true;

optimal_sol:=0;
{*writeln('XIK starts');*}
for i1:=1 to p1 do
    begin
      for k1:=1 to m1 do
          begin
          {*write(XIK[i1,k1]:2:6,' ');*}
            optimal_sol:=optimal_sol + XIK[i1,k1] * modified_BIK[i1,k1];
          end;
   {* writeln;*}
    end;
end;

for i1:=1 to p1 do
    begin
      for k1:=1 to m1 do
          begin
          write(f2,XIK[i1,k1]:2:6);
        end;
      writeln(f2);
    end;

{*writeln('XIK complete');*}
writeln;
write('vam=',optimal_sol:7:2);
writeln;
end;

{****************************************************************************}
```

```
proceDure vam1;

var
      i1,k1,i,j,l,loc,count : integer;
      min,max,temp,sumb,sumd:real;
      difference : array [1..Z] of real;
      all_is_crossed:boolean;
      small : array [1..Z,1..2] of real;
      f3 : text;
begin
   assign(f3,'input_optimal_vamsik.txt');
   rewrite(f3);

  for i1:=1 to p1 do
    begin
       for k1:=1 to m1 do
         begin
                XIK[i1,k1]:=0;
                XIK_not_zero[i1,k1]:=false;
                cross[i1,k1]:=false;
            t[i1,k1]:=0;
         end;
    end;

for i1:=1 to p1 do
    begin
       for k1:=1 to m1 do
         begin
                t[i1,k1]:=modified_vam_input[i1,k1];
         end;
    end;


for i1:=1 to p1 do
    b[i1]:=original_b[i1];
for k1:=1 to m1 do
    d[k1]:=original_d[k1];
optimal_sol:=0;

for i1:=1 to p1 do
    b_boolean[i1]:=true;
for k1:=1 to m1 do
    d_boolean[k1]:=true;

all_is_crossed:=false;
while(all_is_crossed=false) do
begin
{*********making false each time for finding the minima********}
for i1:=1 to p1 do
    begin
        for k1:=1 to m1 do
          begin
            minima[i1,k1]:=false;
          end;
    end;
{*****finding the first and second minimum in a row************}
min:=9999999;
```

```
temp:=-1;
for i1:=1 to p1 do
if (b_boolean[i1]) then
   begin
     for l:=1 to 2 do
       begin
         for loc:=1 to m1 do
           begin
             if
((t[i1,loc]<min)and(minima[i1,loc]=false)and(cross[i1,loc]=false)) then

             begin
             i:=loc;
             min:=t[i1,loc];
             end;
           end;
           minima[i1,i]:=true;
           small[i1,l]:=min;
           temp:=small[i1,1];
           min:=9999999;
       end;
   end
else begin
       small[i1,2]:=-2;
       small[i1,1]:=0;
     end;

{*****finding the first and second minimum in a column*******}
for i:=1 to p1 do
   begin
   for j:=1 to m1 do
       begin
       minima[i,j]:=false;
       end;
   end;

min:=9999999;
temp:=-1;
for k1:=1 to m1 do
if (d_boolean[k1]) then
   begin
     for l:=1 to 2 do
       begin
         for loc:=1 to p1 do
           begin
             if
((t[loc,k1]<min)and(minima[loc,k1]=false)and(cross[loc,k1]=false)) then
             begin
             j:=loc;
             min:=t[loc,k1];
             end;
           end;
           minima[j,k1]:=true;
           small[p1+k1,l]:=min;
           temp:=small[p1+k1,1];
           min:=9999999;
       end;
```

```
    end
else begin
       small[p1+k1,2]:=-2;
       small[p1+k1,1]:=0;
     end;
{****************compute the difference*********************}

for i:=1 to p1+m1 do
    begin
      difference[i]:=small[i,2]-small[i,1];
    end;

{*********finding the maximum of difference*****************}
max.=-0.5;
for i:=1 to p1+m1 do
    begin
      if (difference[i]>max) then
         begin
            max:=difference[i];
            loc:=i;
         end;
    end;

{****if loc is row then finding the cell with minimum cost ***}

if (loc<=p1) then
begin
   i1:=loc;
   min:=9999999;
   for i:=1 to m1 do
     begin
      if ((t[loc,i]<min)and(cross[loc,i]=false)) then
         begin
           l:=i;
           min:=t[loc,l];
         end;
     end;
   k1:=1;
end;

{****if loc is  column then finding the cell with minimum cost ***}

if (loc>p1) then
begin
   min:=9999999;
   k1:=loc-p1;
   for j:=1 to p1 do
     begin
       if ((t[j,k1]<min)and(cross[j,k1]=false)) then
          begin
            l:=j;
            min:=t[l,k1];
          end;
     end;
   i1:=l;
end;
```

```
    if (b[i1]=d[k1]) then
      begin
        XIK[i1,k1]:=d[k1];
        for i:=1 to m1 do
        cross[i1,i]:=true;
        for j:=1 to p1 do
        cross[j,k1]:=true;
        b[i1]:=0;
        d[k1]:=0;
        b_boolean[i1]:= false;
        d_boolean[k1]:= false;
      end;
    if (d[k1]<b[i1]) then
      begin
        XIK[i1,k1]:=d[k1];
        for j:=1 to p1 do
        cross[j,k1]:=true;
        b[i1]:=b[i1]-d[k1];
        d[k1]:=0;
        d_boolean[k1]:=false;
      end;
    if (d[k1]>b[i1]) then
      begin
        XIK[i1,k1]:=b[i1];
        for i:=1 to m1 do
        cross[i1,i]:=true;
        d[k1]:=d[k1]-b[i1];
        b[i1]:=0;
        b_boolean[i1]:=false;
      end;
count:=0;
for i:=1 to p1 do
    begin
      for j:=1 to m1 do
          begin
            if (cross[i,j]=true) then
               begin
               count:=count+1;
               end;
          end;
    end;
{*writeln('count=',count);*}
sumb:=0;
sumd:=0;

for i1:=1 to p1 do
    begin
      sumb:= sumb+b[i1];
      sumd:=sumd+d[i1];
    end;
{*writeln('sumb=',sumb:9:3);
writeln('sumd=',sumd:9:3);
*}
if (count=m1*p1) then all_is_crossed:=true;

optimal_sol:=0;
for i1:=1 to p1 do
```

```
    begin
      for k1:=1 to m1 do
          begin
            optimal_sol:=optimal_sol + XIK[i1,k1] * modified_BIK[i1,k1];
          end;
    end;
end;

for i1:=1 to p1 do
    begin
    for k1:=1 to m1 do
      begin
      write(f3,XIK[i1,k1]:2:6);
        end;
    writeln(f3);
    end;


writeln;
write('vam1=',optimal_sol:7:2);
writeln;
end;

{***************************************************************************}

procedure modified_vam;

var
      i1,k1,i,j,l,loc,count : integer;
      min,max,temp:real;
      difference : array [1..Z] of real;
      all_is_crossed:boolean;
      small : array [1..Z,1..2] of real;
begin

  for i1:=1 to p1 do
    begin
       for k1:=1 to m1 do
         begin
               XIK[i1,k1]:=0;
               XIK_not_zero[i1,k1]:=false;
               cross[i1,k1]:=false;
           t[i1,k1]:=0;
         end;
    end;

 for i1:=1 to p1 do
    begin
       for k1:=1 to m1 do
         begin
               t[i1,k1]:=modified_vam_input[i1,k1];
         end;
    end;


 for i1:=1 to p1 do
    b[i1]:=original_b[i1];
```

```
for k1:=1 to m1 do
    d[k1]:=original_d[k1];
optimal_sol:=0;

all_is_crossed:=false;
while(all_is_crossed=false) do
begin
{*********making false each time for finding the minima********}
for i1:=1 to p1 do
    begin
       for k1:=1 to m1 do
         begin
           minima[i1,k1]:=false;
         end;
    end;
{*****finding the first and second minimum in a row************}
min:=9999999;
temp:=-1;
for i1:=1 to p1 do
   begin
     for l:=1 to 2 do
       begin
         for loc:=1 to m1 do
           begin
             if
((t[i1,loc]<min)and(minima[i1,loc]=false)and(cross[i1,loc]=false)) then

             begin
             i:=loc;
             min:=t[i1,loc];
             end;
           end;
           minima[i1,i]:=true;
           small[i1,l]:=min;
           temp:=small[i1,l];
           min:=9999999;
       end;
   end;

{*****finding the first and second minimum in a column*******}
for i:=1 to p1 do
   begin
   for j:=1 to m1 do
       begin
       minima[i,j]:=false;
       end;
   end;

min:=9999999;
temp:=-1;
for k1:=1 to m1 do
   begin
     for l:=1 to 2 do
       begin
         for loc:=1 to p1 do
           begin
```

```
            if
((t[loc,k1]<min)and(minima[loc,k1]=false)and(cross[loc,k1]=false)) then
            begin
            j:=loc;
            min:=t[loc,k1];
            end;
          end;
          minima[j,k1]:=true;
          small[p1+k1,1]:=min;
          temp:=small[p1+k1,1];
          min:=9999999;
      end;
   end;
{****************compute the difference*********************}

for i:=1 to p1+m1 do
    begin
      difference[i]:=small[i,2]-small[i,1];
    end;

{*********finding the maximum of difference*****************}
max:=0;
for i:=1 to p1+m1 do
    begin
      if (difference[i]>max) then
         begin
            max:=difference[i];
            loc:=i;
         end;
    end;

{****if loc is row then finding the cell with minimum cost ***}

if (loc<=p1) then
begin
   i1:=loc;
   min:=9999999;
   for i:=1 to m1 do
     begin
      if ((t[loc,i]<min)and(cross[loc,i]=false)) then
         begin
           l:=i;
           min:=t[loc,l];
         end;
     end;
   k1:=l;
end;

{****if loc is  column then finding the cell with minimum cost ***}

if (loc>p1) then
begin
   min:=9999999;
   k1:=loc-p1;
   for j:=1 to p1 do
     begin
       if ((t[j,k1]<min)and(cross[j,k1]=false)) then
```

```
          begin
            l:=j;
            min:=t[l,k1];
          end;
     end;
   i1:=l;
end;

     if (b[i1]=d[k1]) then
       begin
         XIK[i1,k1]:=d[k1];
         for i:=1 to m1 do
         cross[i1,i]:=true;
         for j:=1 to p1 do
         cross[j,k1]:=true;
         b[i1]:=0;
         d[k1]:=0;
       end;
     if (d[k1]<b[i1]) then
       begin
         XIK[i1,k1]:=d[k1];
         for j:=1 to p1 do
         cross[j,k1]:=true;
         b[i1]:=b[i1]-d[k1];
         d[k1]:=0;
       end;
     if (d[k1]>b[i1]) then
       begin
         XIK[i1,k1]:=b[i1];
         for i:=1 to m1 do
         cross[i1,i]:=true;
         d[k1]:=d[k1]-b[i1];
         b[i1]:=0;
       end;
count:=0;
for i:=1 to p1 do
    begin
      for j:=1 to m1 do
          begin
            if (cross[i,j]=true) then
               begin
               count:=count+1;
               end;
          end;
    end;
if (count=m1*p1) then all_is_crossed:=true;

optimal_sol:=0;
for i1:=1 to p1 do
    begin
      for k1:=1 to m1 do
          begin
            optimal_sol:=optimal_sol + XIK[i1,k1] * modified_BIK[i1,k1];
          end;
    end;
end;
writeln;
```

```
write('optimal solution saumya=',optimal_sol:7:2);
writeln;
end;

{*************************************************************************}
procedure print_solution;
var
      c1,i1,k1 : integer,

begin {print_solution}
    (*  writeln(p1);
      writeln(m1);
    for i1:=1 to p1 do
      write(original_b[i1]:1:6,' ');
      writeln;
    for k1:=1 to m1 do
      write(original_d[k1]:1:6,' ');
      writeln;*)
      write(obj_fn_value_using_v0_vk:5:2,' =obj val using v0 vk');
      writeln;
      write(No_of_iteration:6,' =no. of iterations');
      writeln;
end; {print_solution}

{*********************************************************************}

BEGIN {MAIN}
      initialization1;
      read_input;
      prepare_set_ifmn_k;
      prepare_min_in_col_array;
      obtain_solution_with_all_zi_at_zero;
      dual_solution_can_be_improved;
      No_of_iteration:=0;
      while (improvement_possible) do
      begin
            No_of_iteration:=No_of_iteration+1;
            determine_extent_of_increase;
            improve_the_solution;
            prepare_set_ifmn_k;
            Prepare_min_in_col_array;
            compute_the_improved_solution;
            dual_solution_can_be_improved;
      end;
      Dual_var_calculation;
      print_solution;
temp_for_saumya;
primal_sol_calculation;
vam;
{*vam(modified_BIK_plus_ZI);*}
vam1;
{*modified_vam;*}
writeln;
END. {MAIN}
```

Appendix 4

tpt vam.c

```
#include<stdio.h>

#define MAX_NODE_NO 102
#define YES 1
#define NO 0
#define ROW 1
#define COL 0
#define INFINITY 10000000
#define SMALL_INFINITY 100000

/*----------Data structures for transportation problem      --------------*/

int count = 0;              /* It counts the no. of nodes in the loop */
int supply_count = 1;       /* Total no of train loads available at all
                         stations */
int iteration_count = 0; /* iterations for optimization in
                         transportation problem */
int basic_count = 0;        /* ?????? */
int found = NO;
int temp[MAX_NODE_NO+1]; /* what context is it used for? */
int row_penalty[MAX_NODE_NO + 2],col_penalty[MAX_NODE_NO+ 2];
int Tot_Row=0,Tot_Col=0;
int sum_demand;

struct node
{
      int if_basic;
      int assigned_value;
      int cost;
      int reduced_cost;
};

struct array_type_for_transportation_problem
{
      struct node node_array[MAX_NODE_NO + 2][MAX_NODE_NO + 2];
      int supply[MAX_NODE_NO + 2];
      int demand[MAX_NODE_NO + 2];
};

struct array_type_for_transportation_problem transport;

struct number_of_basic_variables_in_rows_or_columns
{
      int count;
      int index;
};

struct number_of_basic_variables_in_rows_or_columns temp1,temp2,row,col;
struct outgoing_var
{
      int row_index;
```

```
        int col_index;
  } ;

  struct outgoing_var loop[(MAX_NODE_NO  + 1) * 2];

  struct call_hierarchy
  {
        int received_from_row_index;
        int received_from_col_index;
  };

  struct call_hierarchy the_immidiate_caller[MAX_NODE_NO + 2][MAX_NODE_NO+ 2];

  struct link{
        int node1;
        int node2;
  };

  //void basic_feasible_sol_by_NW_corner(int row_size, int col_size,
  //        int row_index, int col_index);
  void get_row_and_col_penalties(int row_size, int col_size, int type,
         int max_allocation_index, int calling_index);
  void reduced_cost_function(int row_size, int col_size);
  void optimality_test(int row_size, int col_size);
  void get_improved_solution(int row_size, int col_size,
         int entering_row_index, int entering_col_index, int count);
  void find_cycle(int row_size, int col_size, int row_index,int col_index,
          int look_up_path, int if_first_call, int start_row_index,
          int start_col_index);
  void print_the_whole_path(int start_row_index, int start_col_index,
              int print_this_row_index, int print_this_col_index);
  void print_BFS(int row_size,int col_size);
  void get_new_assigned_values(int row_size, int col_size,
         int outgoing_row_index, int outgoing_col_index, int size);
  void optimum_transportation_cost(int row_size, int col_size);

  /*----------- End of data structure for transportation --------------*/
  /*
  void print_BFS(int row_size,int col_size){
        int i,j,basic_count = 0;
        FILE *fptr = fopen("BFS","a");
        //FILE *pt3;
          int result_tpt[201][201];

        for(i=1;i<row_size;i++){
              for(j=1;j<col_size;j++){
                    if(transport.node_array[i][j].if_basic == YES){
                          basic_count++;
                          fprintf(fptr,"x[%d][%d] =
  %d\n",i,j,transport.node_array[i][j].assigned_value);
                                      //here

  //result_tpt[j][i]=transport.node_array[i][j].assigned_value;// skewed reading
  of result

                    }
              }
```

```
        }
        fprintf(fptr,"Basic Count = %d\n",basic_count);
        fclose(fptr);

            pt3=fopen("result_tpt.txt","w");

            for(i=1;i<=row_size;i++)
            { for(j=1;j<=col_size;j++)
                 { fprintf(pt3,"%d\t",result_tpt[i][j]) ;
                 }
                 fprintf(pt3,"\n");
            }
            close(pt3);

}
*/

/*Reading the input Basic Feasible Solution*/

void starting_good_BFS(int row_size, int col_size)
{
     int i=0,j=0;
     int x[row_size][col_size];
     float temp=0,temp2=0;
     int temp1=0;

     FILE *pt4;

     pt4=fopen("input_optimal_vam.txt","r");

     for(i=1;i<=Tot_Row;i++)
     { for(j=1;j<=Tot_Col;j++)
       { x[i][j]=0;
       }
     }

 //    printf("\nsum demand=%d\n",sum_demand);


     for(i=1;i<=Tot_Row;i++)
     { for(j=1;j<=Tot_Col;j++)
       { fscanf(pt4,"%f",&temp);         //This is to read the BFS
       temp2=temp*(float)sum_demand*10000;
       temp1=(int)temp2;
         x[i][j]=temp1;//already in the skewed form.
       }
     }
    close(pt4);

/*
    for(i=1;i<=Tot_Row;i++)
     { for(j=1;j<=Tot_Col;j++)
       { printf("%d\t",x[i][j]);
       }
        printf("\n");
     }
*/
```

```
    for(i=1;i<=Tot_Row;i++)
    { for(j=1;j<=Tot_Col;j++)
      { if( x[i][j]!=0)
            { transport.node_array[i][j].assigned_value=x[i][j];
              transport.node_array[i][j].if_basic=YES;
            //  printf("x[%d][%d]=%d\n",i,j,
transport.node_array[i][j].assigned_value);
            }
      }
    }

    transport.node_array[MAX_NODE_NO+1][MAX_NODE_NO+1].assigned_value=0;//CHECK
it
    transport.node_array[MAX_NODE_NO+1][MAX_NODE_NO+1].if_basic=YES;
}


/*--------------------- transportation() ------------------------*/
int k1_if_basic = 0;
void transportation()          /* called in F_Decision_Point_Arrives */
{
      int i, j, k, train_no, supply_count;
      int total_demand = 0, total_supply = 0;
      int max_row_no=Tot_Row;
      int max_col_no=Tot_Col;
      //FILE *fp = fopen("check ","a");
//commentprintf("inside transportation\n??????????????????????????\n");
/*printf("\n\nINITIAL BASIC FEASIBLE SOLUTION\n");
//commentprintf("................................\n");   */

      for(i = 1;i < max_row_no + 1;i++)
            total_demand += transport.demand[i];

      for(j = 1;j < max_col_no + 1;j++)
            total_supply += transport.supply[j];
/*
      if(total_supply != total_demand) {
            if(total_demand < total_supply) {
                  transport.demand[max_row_no + 1] = total_supply
                                                    - total_demand;
                  for(j = 1;j < max_col_no + 1;j++) {
                        transport.node_array[max_row_no + 1][j].if_basic=0;
                        transport.node_array[max_row_no + 1][j].cost = 0;
                  }

                  basic_feasible_sol_by_NW_corner(max_row_no + 2,
                                          max_col_no + 1, 1, 1);
                  print_BFS(max_row_no + 2,max_col_no + 1);
                  //printf("k1_if_basic = %d\n",k1_if_basic);
                  optimality_test(max_row_no + 2, max_col_no + 1);
            }


                  else{
                  transport.supply[max_col_no + 1] = total_demand
                                                    - total_supply;
```

```
                for(i = 1;i < max_row_no + 1;i++) {
                        transport.node_array[i][max_col_no + 1].if_basic=0;
                        transport.node_array[i][max_col_no + 1].cost = 0;
                }
                basic_feasible_sol_by_NW_corner(max_row_no + 1,
                                                max_col_no + 2, 1, 1);
                print_BFS(max_row_no + 1,max_col_no + 2);
                //printf("k1_if_basic = %d\n",k1_if_basic);
                optimality_test(max_row_no + 1, max_col_no + 2);
            }
        }
        else
*/
         {
                //basic_feasible_sol_by_NW_corner(max_row_no + 1,
            //                                max_col_no + 1, 1, 1);
            //print_BFS(max_row_no + 1,max_col_no + 1);
            //printf("k1_if_basic = %d\n",k1_if_basic);

                starting_good_BFS(max_row_no + 1,max_col_no + 1);//here
            optimality_test(max_row_no + 1, max_col_no + 1);
        }
        //fclose(fp);
}
/*--------------- basic_feasible_sol_by_NW_corner() ---------------*/

void basic_feasible_sol_by_NW_corner(int row_size, int col_size, int row_index,
int col_index)
{
        int i,j;
        //FILE *fp = fopen("check ","a");

          //fprintf(fp,"row_size = %d\ncol_size = %d\n",row_size,col_size);
        if(row_index == row_size || col_index ==  col_size)
            return;


        if(transport.demand[row_index] >= transport.supply[col_index])
        {
            transport.node_array[row_index][col_index].assigned_value =
transport.supply[col_index];
            //printf("transport.supply[%d] =
%d\n",col_index,transport.supply[col_index]);
            transport.demand[row_index] = transport.demand[row_index] -
transport.supply[col_index];
            transport.supply[col_index] = 0;
            transport.node_array[row_index][col_index].if_basic = YES;
            k1_if_basic++;
            if(col_index+1 == col_size){
                if(basic_count != (row_size + col_size - 3)){
                        for(i=row_index;i<row_size;i++){
                                transport.node_array[i][col_index].if_basic = YES;
                                basic_count++;
                                transport.node_array[i][col_index].assigned_value
= 0;

                        }
                }
```

```
            }
        /*printf("node[%d][%d].assigned_value = %d\nnode[%d][%d].if_basic = %d\n",
row_index, col_index, transport.node_array
            [row_index][col_index].assigned_value, row_index, col_index,
transport.node_array[row_index][col_index].if_basic);*/
            basic_feasible_sol_by_NW_corner(row_size, col_size, row_index,
col_index+1);
        }
        else
        {
            transport.node_array[row_index][col_index].assigned_value =
transport.demand[row_index];
            //printf("transport.demand[%d] =
%d\n",row_index,transport.demand[row_index]);
            transport.supply[col_index] = transport.supply[col_index] -
transport.demand[row_index];
            transport.demand[row_index] = 0;
            transport.node_array[row_index][col_index].if_basic = YES;
            k1_if_basic++;
            if(row_index + 1 == row_size){
                if(basic_count != (row_size + col_size -3)){
                    for(j = col_index;j < col_size;j++){
                        transport.node_array[row_index][j].if_basic = YES;
                        basic_count++;
                        transport.node_array[row_index][j].assigned_value
= 0;
                    }
                }
            }
            /*fprintf(fp,"k1_if_basic = %d\n",k1_if_basic);
            printf("node[%d][%d].assigned_value = %d\nnode[%d][%d].if_basic =
%d\n", row_index, col_index, transport.node_array
            [row_index][col_index].assigned_value, row_index, col_index,
transport.node_array[row_index][col_index].if_basic);*/
            basic_feasible_sol_by_NW_corner(row_size, col_size, row_index + 1,
col_index);
        }
        //fclose(fp);
}

/*-------------- optimality_test() ---------------------------*/

void optimality_test(int row_size, int col_size)
{
        int i, j;
        FILE *fp ;
        templ.count = 0;
        //fp = fopen("check","a");

        //fprintf(fp,"row_size = %d\ncol_size = %d\n",row_size,col_size);

        for(i = 1;i < row_size;i++) {
            row.count = 0;
            row.index = i;

            for(j = 1;j < col_size;j++) {
              if(transport.node_array[i][j].if_basic == YES)
```

```
                row.count = row.count + 1;
            }

            if(row.count > temp1.count) {
              temp1.count = row.count;
              temp1.index = row.index;
            }
      }
      temp2.count = 0;

      for(j = 1;j < col_size;j++) {
            col.count = 0;
            col.index = j;

            for(i=1;i<row_size;i = i+1){

                 if(transport.node_array[i][j].if_basic == YES)
                     col.count = col.count + 1;
            }
            if(col.count > temp2.count){
                  temp2.count = col.count;
                  temp2.index = col.index;
            }
      }

      //printf("\n");
      for(i = 1;i < row_size;i++)
                  for(j = 1;j < col_size;j++)
                        if(transport.node_array[i][j].if_basic == YES){
                              basic_count++;
                        //commentprintf("[%d][%d]\n", i, j);
                        }
      // perror("\n I am before basic_count");
      //printf("basic_count = %d\n", basic_count);

      if(temp1.count >= temp2.count) {
            row_penalty[temp1.index] = 0;
            /*printf("first\n");
            print_matrix(row_size, col_size);
      //commentprintf("penalty for row[%d] found \n", temp1.index);
            penalty_count = 1;*/
            get_row_and_col_penalties(row_size, col_size, ROW, temp1.index, -1);
            /*printf("out\n");
            for(i = 1;i < row_size;i++)
            //commentprintf("row_penalty[%d] = %d\n", i, row_penalty[i]);
            for(j = 1;j < col_size;j++)
            //commentprintf("col_penalty[%d] = %d\n", j, col_penalty[j]);*/
      }

       else {
            col_penalty[temp2.index] = 0;
            /*printf("second\n");
            print_matrix(row_size, col_size);
       //commentprintf("penalty for col[%d] found \n", temp2.index);
            penalty_count = 1;*/
            get_row_and_col_penalties(row_size, col_size, COL, temp2.index , -
1);
```

```
                /*printf("out\n");
                for(i = 1;i < row_size;i++)
                //commentprintf("row_penalty[%d] = %d\n", i, row_penalty[i]);
                for(j = 1;j < col_size;j++)
                //commentprintf("col_penalty[%d] = %d\n", j, col_penalty[j]);*/
          }
          /*for(i=1;i<row_size;i++)
          //commentprintf("row_penalty[%d] =%d\n", i, row_penalty[i]);
      //commentprintf("\n");
          for(j=1;j<col_size;j++)
          //commentprintf("col_penalty[%d] =%d\n", j, col_penalty[j]);
           //commentprintf("\n");
           //commentprintf("calling reduced_cost\n");*/
          reduced_cost_function(row_size, col_size);
}
/*---------------------- get_row_and_col_penalties() ----------------*/

void get_row_and_col_penalties(int row_size, int col_size, int type, int
max_allocation_index, int calling_index)
{
          int i, j;
/*if(penalty_count != row_size + col_size - 2) return;
//commentprintf("inside
get_row_and_col_penalties\n????????????????????????????????\n");*/
          if(type == ROW)
          {
            for(j = 1;j < col_size;j++)
            {
                if(j == calling_index)
                continue;

                if(transport.node_array[max_allocation_index][j].if_basic == YES)
                {
                //printf("penalty for col[%d] found \n", j);
                //penalty_count++;
                col_penalty[j] = transport.node_array[max_allocation_index][j].cost
- row_penalty[max_allocation_index] ;
                //if(penalty_count != row_size + col_size - 2)
                get_row_and_col_penalties(row_size, col_size, COL, j,
max_allocation_index);
                //else return;
                }
            }
          }

          if(type == COL)
          {
            for(i = 1;i < row_size;i++)
            {
               if(i == calling_index)
                continue;

               if(transport.node_array[i][max_allocation_index].if_basic == YES)
               {
                //printf("penalty for row[%d] found\n", i);
                ///penalty_count++;
```

```
            row_penalty[i] = transport.node_array[i][max_allocation_index].cost
- col_penalty[max_allocation_index];
            //if(penalty_count != row_size + col_size - 2)
            get_row_and_col_penalties(row_size, col_size, ROW, i,
max_allocation_index);
            //else return;
           }
        }
       }

return;
}

void reduced_cost_function(int row_size, int col_size)
{
      int i, j;
      int temp = 0;
      int entering_row_index = -1;
      int entering_col_index = -1;

      for(i = 1;i < row_size;i++)
      {
         for(j = 1;j < col_size;j++)
         {
            //if( transport.node_array[i][j].if_basic == YES)
            //continue;
            //else{
            transport.node_array[i][j].reduced_cost = row_penalty[i] +
col_penalty[j] - transport.node_array[i][j].cost;

                  //printf("reduced_cost[%d][%d]=%d\n", i, j,
transport.node_array[i][j].reduced_cost);
            //}
            if(temp < transport.node_array[i][j].reduced_cost)
            {
                  temp = transport.node_array[i][j].reduced_cost;
                  entering_row_index = i;
                  entering_col_index = j;

            }
         }
      }

   /*for(i = 1;i < row_size;i++)
      for(j = 1;j < col_size;j++)
            if(transport.node_array[i][j].reduced_cost >= 0)
            //commentprintf("transport.node_array[%d][%d].reduced_cost = %d\n",
i, j, transport.node_array[i][j].reduced_cost);*/
      if(entering_row_index == -1)
      {
      optimum_transportation_cost(row_size, col_size);
      /*printf("inside\n");
       //commentprintf("\n\npresent solution is OPTIMAL solution of
TRANSPORTATOION problem\n");
      for(i=1;i<row_size;i++)
        for(j=1;j<col_size;j++)
           if(transport.node_array[i][j].if_basic==YES)
```

```
            //commentprintf("\n[%d][%d].assigned_value=%d\n", i, j,
transport.node_array[i][j].assigned_value);
      optimum_transportation_cost(row_size, col_size);
      //commentprintf("\nOPTIMAL COST=%d\n",
optimum_transportation_cost(row_size, col_size));
   //commentprintf("\nITERATIIONS USED =%d\n",  iteration_count);*/
        printf("\nITERATIIONS USED IN VAM=%d\n",  iteration_count);//here
      found = YES;
      }

      else
      {
        //printf("\nEntering node =node[%d][%d]\n\n", entering_row_index,
entering_col_index);
        basic_count = 0;
        get_improved_solution(row_size, col_size, entering_row_index,
entering_col_index, count);
      }
}

void get_improved_solution(int row_size, int col_size, int entering_row_index,
int entering_col_index, int count)
{
      int temp = 0, i;
      int start_row_index, start_col_index;
      start_row_index = entering_row_index;
      start_col_index = entering_col_index;
      find_cycle(row_size, col_size, entering_row_index, entering_col_index,
ROW, YES, start_row_index, start_col_index);
}

void find_cycle(int row_size, int col_size, int row_index, int col_index, int
look_up_path, int if_first_call, int start_row_index,
int start_col_index )
{
      int i, j, k;
      int temp = 0;
      transport.node_array[start_row_index][start_col_index].if_basic = YES;
      transport.node_array[start_row_index][start_col_index].assigned_value = 0;

      if(row_index == -1 && col_index == -1)
            return;

      if(look_up_path == ROW)
      {
            for(j = 1;j < col_size;j++)
            {
                  if(transport.node_array[row_index][j].if_basic == YES)
                  {
                        if(found == YES)
                              break;
                        if(j == col_index)
                              continue;

      the_immidiate_caller[row_index][j].received_from_row_index = row_index;
```

```
      the_immidiate_caller[row_index][j].received_from_col_index = col_index;
                                       if(row_index == start_row_index && j ==
start_col_index)
                              {/*where does this close*/
                               /****/count = 0;

                               print_the_whole_path(start_row_index, start_col_index,
the_immidiate_caller[row_index][j].
                               received_from_row_index,
the_immidiate_caller[row_index][j].received_from_col_index);

                               for(k = 0;k < count - 1;k++)
                               {
                                   /*printf("loop[%d][%d].assigned_value = %d\n",
loop[k].row_index, loop[k].col_index,

      transport.node_array[loop[k].row_index][loop[k].col_index].assigned_value)
;*/
                                     if((k%2) != 0)
                                           continue;
                                     else
                                     {

if(transport.node_array[loop[k].row_index][loop[k].col_index].assigned_value
                                          <
transport.node_array[loop[temp].row_index][loop[temp].col_index].assigned_value)
                                         temp = k;
                                     }
                               }
                              //printf("\noutgoing_node = NOde[%d][%d]\n",
loop[temp].row_index, loop[temp].col_index);

transport.node_array[loop[temp].row_index][loop[temp].col_index].if_basic = NO;
                              //printf("\nIteration_no=%d\n", iteration_count + 1);
                               get_new_assigned_values(row_size, col_size,
loop[temp].row_index, loop[temp].col_index, count);
                              }
                              else
                                   find_cycle(row_size, col_size, row_index, j, COL,
NO, start_row_index, start_col_index);
                        }
                        if(found == YES)
                              break;
                 }
      }

      if(look_up_path == COL)
      {
            for(i = 1;i < row_size;i++)
            {
                  if(transport.node_array[i][col_index].if_basic == YES)
                  {
                        if(found == YES)
                              break;
                        if(i == row_index)
                              continue;
```

```
      the_immidiate_caller[i][col_index].received_from_row_index = row_index;

      the_immidiate_caller[i][col_index].received_from_col_index = col_index;
                            if(i == start_row_index && col_index == start_col_index)
                            {
                               count = 0;
                               print_the_whole_path(start_row_index,
start_col_index, the_immidiate_caller[i][col_index].
                               received_from_row_index,
the_immidiate_caller[i][col_index].received_from_col_index);

                               for(k = 0;k < count - 1;k++)
                               {
                                  /*printf("loop[%d][%d].assigned_value = %d\n",
loop[k].row_index, loop[k].col_index,

      transport.node_array[loop[k].row_index][loop[k].col_index].assigned_value)
;*/
                                  if((k%2) != 0)
                                       continue;
                                  else
                                  {

if(transport.node_array[loop[k].row_index][loop[k].col_index].assigned_value
                                       <
transport.node_array[loop[temp].row_index][loop[temp].col_index].assigned_value)
                                       temp = k;
                                  }
                               }

                            //printf("\noutgoing_node = NOde[%d][%d]\n",
loop[temp].row_index, loop[temp].col_index);

transport.node_array[loop[temp].row_index][loop[temp].col_index].if_basic = NO;
                              // printf("\nIteration_no=%d\n", iteration_count + 1);
                               get_new_assigned_values(row_size, col_size,
loop[temp].row_index, loop[temp].col_index, count);
                            }
                            else
                               find_cycle(row_size, col_size, i, col_index, ROW, NO,
start_row_index, start_col_index);
                       }
                       if(found == YES)
                            break;
              }
      }
}


void print_the_whole_path(int start_row_index, int start_col_index, int
print_this_row_index, int print_this_col_index)
{
      //printf("cycle_node[%d][%d]\n", print_this_row_index,
print_this_col_index);
      loop[count].row_index = print_this_row_index;
      loop[count].col_index = print_this_col_index;
```

```
      count = count+1;
      if((print_this_row_index == start_row_index && print_this_col_index ==
start_col_index) || found == YES)
            return;
      print_the_whole_path(start_row_index, start_col_index,
the_immidiate_caller[print_this_row_index][print_this_col_index]
      .received_from_row_index,
the_immidiate_caller[print_this_row_index][print_this_col_index].received_from_c
ol_index);
}

void get_new_assigned_values(int row_size, int col_size, int outgoing_row_index,
int outgoing_col_index, int size)
{
      int i, j;
      for(i = 0;i < size;i++)
      {
          if(loop[i].row_index == outgoing_row_index && loop[i].col_index ==
outgoing_col_index)
          {
            transport.node_array[loop[i].row_index][loop[i].col_index].if_basic
= NO;
                  continue;
          }

          if((i%2) == 0)
          {
            transport.node_array[loop[i].row_index][loop[i].col_index].if_basic
= YES;

      transport.node_array[loop[i].row_index][loop[i].col_index].assigned_value
-= transport.node_array
            [outgoing_row_index][outgoing_col_index].assigned_value;
          }

          else
          {
            transport.node_array[loop[i].row_index][loop[i].col_index].if_basic
= YES;

      transport.node_array[loop[i].row_index][loop[i].col_index].assigned_value
+= transport.node_array
            [outgoing_row_index][outgoing_col_index].assigned_value;
          }
      }

      transport.node_array[outgoing_row_index][outgoing_col_index].assigned_valu
e = 0;

      /*printf("\n");
      for(i=1;i<row_size;i++)
            for(j = 1;j < col_size;j++)
              //commentprintf("assigned[%d][%d] = %d\nif_basic[%d][%d] =
%d\n\n", i, j, transport.node_array[i][j]
                  .assigned_value, i, j, transport.node_array[i][j].if_basic)*/
      iteration_count++;
      optimality_test(row_size, col_size);
```

```
}
/*--------------- optimum_transportation_cost() ---------------*/

void optimum_transportation_cost(int row_size, int col_size)
{
  int i, j, sum = 0;
  for(i = 1;i < row_size;i++)
      for(j = 1;j < col_size;j++)
      {

        if(transport.node_array[i][j].if_basic == YES){
         // printf("cost[%d][%d]=%d\n", i, j, transport.node_array[i][j].cost);
         // printf("assigned_value[%d][%d]=%d\n", i, j,
transport.node_array[i][j].assigned_value);
             sum +=
transport.node_array[i][j].assigned_value*transport.node_array[i][j].cost;
                    }
      }
  sum=sum/10000;
  printf("\nOptimal Cost VAM =%d\n",sum);//here
}

/*----------------------- main() ------------------------------*/

main()
{
      int i, j,t, start_node, if_scheduled = NO, no_of_rains_having_no_destin;
      int time1, time_index, node_index, train_no, curr_stn;
      int quantity_to_transport,rake_type, goods_type;
        //int Tot_Row=0,Tot_Col=0;
        int demand_gen[201],supply_gen[201];
        int cost_gen[201][201];
        int temp=0;

      FILE *pt1;
        FILE *pt2;

        pt2=fopen("dmd_spl.txt","r");
        pt1=fopen("cost.txt","r");

        fscanf(pt2,"%d",&Tot_Row);
        fscanf(pt2,"%d",&Tot_Col);

   for(t=1;t<=Tot_Row;t++)
   {    for(i=1;i<=Tot_Col;i++)
        { fscanf(pt1,"%d",&temp);        //This is to read the transpose
transport costs
          cost_gen[i][t]=temp;
        }
   }

   for(t=1;t<=Tot_Row;t++)
   {    fscanf(pt2,"%d",&supply_gen[t]);    //This is to read the initial
supplies
   }
   for(t=1;t<=Tot_Col;t++)
```

```
   {     fscanf(pt2,"%d",&demand_gen[t]);     //This is to read the initial
demands
   }

   close(pt2);  // pt2 reads value of Tot_Row, Tot_Col from 1st row of
dmd_spl.txt
         // pt2: supply from 2nd row & demand from 3rd row rsptly from
dmd_spl.txt

   close(pt1);     // pt reads cost matrix

   for(i=1;i<=Tot_Row;i++)
   { sum_demand=sum_demand+demand_gen[i];
   }
   //printf("\nsum demand in main=%d\n",sum_demand);

      for(i = 1;i <= Tot_Row;i++)  {
            transport.supply[i] = supply_gen[i];
            transport.demand[i] = demand_gen[i];
      }

      for(i=1;i<=Tot_Row;i++)
            for(j=1;j<=Tot_Col;j++){
                  transport.node_array[i][j].if_basic = 0;
                        transport.node_array[i][j].assigned_value=0;//here
                  if(i == j)
                        transport.node_array[i][j].cost = cost_gen[i][j];
                  else {transport.node_array[i][j].cost = cost_gen[i][j];  }
            }

      transportation();
}
```

15

Appendix 5

tpt vamsik.c

Put these two functions on the place of the corresponding functions in tpt_vam.c

```
/*Reading the input Basic Feasible Solution*/

void starting_good_BFS(int row_size, int col_size)
{
      int i=0,j=0;
      int x[row_size][col_size];
      float temp=0,temp2=0;
      int temp1=0;

      FILE *pt4;

      pt4=fopen("input_optimal_vamsik.txt","r");

      for(i=1;i<=Tot_Row;i++)
      { for(j=1;j<=Tot_Col;j++)
        { x[i][j]=0;
        }
      }

      //printf("\nsum demand=%d\n",sum_demand);


      for(i=1;i<=Tot_Row;i++)
      { for(j=1;j<=Tot_Col;j++)
        { fscanf(pt4,"%f",&temp);        //This is to read the BFS
        temp2=temp*(float)sum_demand*10000;
        temp1=(int)temp2;
          x[i][j]=temp1;//already in the skewed form.
        }
      }
     close(pt4);

/*
     for(i=1;i<=Tot_Row;i++)
      { for(j=1;j<=Tot_Col;j++)
        { printf("%d\t",x[i][j]);
        }
         printf("\n");
      }
*/

     for(i=1;i<=Tot_Row;i++)
     { for(j=1;j<=Tot_Col;j++)
       { if( x[i][j]!=0)
            { transport.node_array[i][j].assigned_value=x[i][j];
              transport.node_array[i][j].if_basic=YES;
```

```
          //  printf("x[%d][%d]=%d\n",i,j,
transport.node_array[i][j].assigned_value);
          }
      }
    }


transport.node_array[MAX_NODE_NO+1][MAX_NODE_NO+1].assigned_value=0;//C
HECK it
    transport.node_array[MAX_NODE_NO+1][MAX_NODE_NO+1].if_basic=YES;
}
```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```
/*--------------- optimum_transportation_cost() ---------------*/

void optimum_transportation_cost(int row_size, int col_size)
{
  int i, j, sum = 0;
  for(i = 1;i < row_size;i++)
      for(j = 1;j < col_size;j++)
      {

        if(transport.node_array[i][j].if_basic == YES){
         // printf("cost[%d][%d]=%d\n", i, j,
transport.node_array[i][j].cost);
         // printf("assigned_value[%d][%d]=%d\n", i, j,
transport.node_array[i][j].assigned_value);
            sum +=
transport.node_array[i][j].assigned_value*transport.node_array[i][j].co
st;
                        }
      }
  sum=sum/10000;
  printf("\nOptimal Cost VAM using Sik =%d\n",sum);//here
}
```

Appendix 6

Pascal to C converted code of K.D. Sharma[4]

```
/*Heuristic_for_STP*/
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define I 105
#define K 105
#define true 1
#define false 0
 //int I,K;//I=no. of plants,K=no of markets
 int c2=0,i1,k1,No_of_iteration=0,count=0;
 int ifmn[K+1][I+1];
 int b[I+1];
 int d[I+1];
 float original_BIK[K+1][I+1],BIK_plus_zi[K+1][I+1];
 float B3K[K+1],B1K[K+1],B2K[K+1];
 int z_for_increase[I+1],k_for_increase[K+1];
 int p1=0,m1=0;
 float value_of_zi[I+1];
 float obj_fn_value_using_v0_vk,v0;
 float value_of_vk[K+1];
 int collection_of_zi_that_can_be_increased[I+1];
 int min_in_col[K+1][I+1];
 int single_zi_that_can_be_increased;
 float objective_fn_value, current_loss, cumulative_loss, extent_of_increase;
 int improvement_possible;
 int result;

  int z_i[I+1];
  int union_of_ifmn_sets[I+1];
  int group_set[I+1];
  int k_set_for_group_set[I+1];
  int union_of_ifmn_sets_group_set[I+1];

int round(float temp)
{   int rtn;
    if( (fabs(temp)-temp) >= 0.5 )
       rtn=(int)fabs(temp+1);
    else
       rtn=(int)fabs(temp);
}


void initialization()

{
int index;
int K1,I1;

FILE *ptr;

      ptr=fopen("dmd_spl.txt","r");
```

```
        fscanf(ptr,"%d",&K1);
        fscanf(ptr,"%d",&I1);
close(ptr);

  { for(index=1;index<=K1;index++)
        { for(i1=0;i1<=I1;i1++)
            { ifmn[index][i1]=0;
              printf("%d\t",ifmn[index][i1]);
            }
          printf("\n");
        }
  }
  printf("\n\n\n");
  { for(index=1;index<=I1;index++)
        { value_of_zi[index]=0;
          printf("%d\t",value_of_zi[index]);
        }
  }
  printf("\n\n\n");
  { for(index=1;index<=I1;index++)
        { B1K[index]=0;
          B2K[index]=0;
        }
  }
  { for(index=1;index<=I1;index++)
        { printf("%d\t",B1K[index]);
        }
  }
  printf("\n\n\n");
  { for(index=1;index<=I1;index++)
        { printf("%d\t",B2K[index]);
        }
  }
  printf("\n\n\n");
  cumulative_loss=0;
  current_loss=0;
  extent_of_increase=0;
  printf("cumulative_loss= %f\n",cumulative_loss);
  printf("current_loss= %f\n",current_loss);
  printf("extent_of_increase= %f\n",extent_of_increase);

}//end of void



void read_input()
{
      int temp=0;
      FILE *pt;
      FILE *pt1;
      int t,i;

      pt=fopen("dmd_spl.txt","r");
      pt1=fopen("cost.txt","r");
      //printf("Enter the value of m1, i.e. no. of rows:");
      fscanf(pt,"%d",&m1);
      //printf("Enter the value of p1, i.e. no. of columns:");
```

```
        fscanf(pt,"%d",&p1);

        for(i1=1;i1<=p1;i1++)
            {fscanf(pt,"%d",&b[i1]);          //b[p1]=supply matrix
            }

        for(k1=1;k1<=m1;k1++)
            {fscanf(pt,"%d",&d[k1]);          //d[m1]=demand matrix
            }
        close(pt);

        { for(i=0;i<=p1;i++)
                { printf("%d\t",b[i]);
                }
        }
        printf("\n");

        { for(i=0;i<=m1;i++)
                { printf("%d\t",d[i]);
                }
        }
        printf("\n");

        for(k1=1;k1<=m1;k1++)
            {
             for(i1=1;i1<=p1;i1++)
                {
                    fscanf(pt1,"%d",&temp);
                    original_BIK[i1][k1]=temp;//we'll read skewed cost matrix here
                }
          }
         close(pt1);

         for(t=0;t<=m1;t++)
          { for(i=0;i<=p1;i++)
                { printf("%f\t",original_BIK[t][i]);
                }
            printf("\n");
          }


         for(t=0;t<=m1;t++)
        { for(i=0;i<=p1;i++)
              BIK_plus_zi[t][i]=original_BIK[t][i];
        }

}//end of void


void prepare_set_ifmn_k()

{
   printf("prepare set ifmn k started\n");

     for(k1=1;k1<=m1;k1++)
          { for(i1=1;i1<=p1;i1++)
```

```
                {ifmn[k1][0]=0;
                 ifmn[k1][i1]=0;
                }
          }
      printf("prepare set ifmn k complete1\n");

     for(k1=1;k1<=m1;k1++)
          { B1K[k1]=100000000;
            B2K[k1]=9999999;
            ifmn[k1][0]=0;
            for(i1=1;i1<=p1;i1++)
               { if (BIK_plus_zi[k1][i1]<B1K[k1])
                    { ifmn[k1][0]=1;
                      ifmn[k1][1]=i1;
                      B2K[k1]=B1K[k1];
                      B1K[k1]=BIK_plus_zi[k1][i1];
                    }
                 else if (BIK_plus_zi[k1][i1]=B1K[k1])
                    { ifmn[k1][0]=ifmn[k1][0]+1;
                      ifmn[k1][ifmn[k1][0]]=i1;
                    }
                 else if ((BIK_plus_zi[k1][i1]>B1K[k1]) &&
(BIK_plus_zi[k1][i1]<B2K[k1]))
                    { B2K[k1]=BIK_plus_zi[k1][i1];
                    }
               }//end of for i1
          } //end of for k1
     printf("prepare set ifmn k complete\n");
}//end of void


void obtain_solution_with_all_zi_at_zero()

{
    for(i1=1;i1<=p1;i1++)
        { value_of_zi[i1]=0;
      }

   objective_fn_value=0;
   for(k1=1;k1<=m1;k1++)
        { objective_fn_value=objective_fn_value+B1K[k1]*d[k1];
      }
   printf("obtain_solution_with_all_zi_at_zero complete\n");
}


void prepare_min_in_col_array()

{
 int col_no,row_no;
    for(i1=1;i1<=p1;i1++)
        { min_in_col[0][i1]=0;
            {for(k1=1;k1<=m1;k1++)
                { min_in_col[k1][i1]=0;
                }
            }
```

```
        }

     for(k1=1;k1<=m1;k1++)
          { row_no=ifmn[k1][0];
               { for(i1=1;i1<=row_no;i1++)
                    { col_no=ifmn[k1][i1];
                      min_in_col[0][col_no]=min_in_col[0][col_no]+1;
                      min_in_col[min_in_col[0][col_no]][col_no]=k1;
                    }
               }
          }
    printf("end of  prepare_min_in_col_array\n");
} //end of void

/*solution improvement using sets starts*/

void k1_in_k_set_for_group_set(int k1)
{
      int i=0,j=0,k=0;
      result=0;
      i=1;
      j=k_set_for_group_set[0];
      k=(i+j)/2;

      while((k_set_for_group_set[k]!=k1) || (i<j))
      { if(k1>k_set_for_group_set[k])
          i=k+1;
        else
            j=k-1;
            k=(i+j)/2;
      }

      if(i>j)
        result=0;
      else
        result=1;
      return;
}


void i1_in_union_of_ifmn_sets_group_set(int i1)
{
      int i=0,j=0,k=0;
      result=0;
      i=1;
      j=union_of_ifmn_sets_group_set[0];
      k=(i+j)/2;

      while((union_of_ifmn_sets_group_set[k]!=k1) || (i<j))
      { if(k1>union_of_ifmn_sets_group_set[k])
          i=k+1;
        else
            j=k-1;
            k=(i+j)/2;
      }

      if(i>j)
```

```
        result=0;
      else
        result=0;
      return;
}

void ifmn_in_group_set(int k1)
{
      int i=0,j=0,k=0;
      result=0;
      i=1;
      j=group_set[0];
      k=(i+j)/2;

      while((group_set[k]!=k1) || (i<j))
      { if(k1>group_set[k])
          i=k+1;
        else
            j=k-1;
            k=(i+j)/2;
      }

      if(i>j)
        result=0;
      else
        result=1;
      return;
}


void i1_in_group_set(int k1)
{
      int i=0,j=0,k=0;
      result=0;
      i=1;
      j=group_set[0];
      k=(i+j)/2;

      while((group_set[k]!=k1) || (i<j))
      { if(k1>group_set[k])
          i=k+1;
        else
            j=k-1;
            k=(i+j)/2;
      }

      if(i>j)
        result=0;
      else
        result=1;
      return;
}


void add_to_union_of_ifmn_sets(int k1)
{
      int i=0,j=0,t=0;
```

```
int found;

i=1;
j=union_of_ifmn_sets[0];
found=0;
for(i=1;i<=j;i++)
{ if( k1=union_of_ifmn_sets[i] )
    found=1;
}

if(found=1) return;
else
   { if(j=0) union_of_ifmn_sets[1]=k1;
     else
         { if(k1>union_of_ifmn_sets[j])
              union_of_ifmn_sets[j+1]=k1;
           else
               {i=1;
                while(k1>union_of_ifmn_sets[i])
                     {i=i+1;
                     }
                for(t=j;t>=i;t--)
                   { union_of_ifmn_sets[j+1]=union_of_ifmn_sets[t];
                   }
                union_of_ifmn_sets[i]=k1;
               }
         }
   }
}


void add_to_group_set(int k1)
{
    int i=0,j=0,t=0;
    int found;

    i=1;
    j=group_set[0];
    found=0;
    for(i=1;i<=j;i++)
    { if( k1=group_set[i] )
        found=1;
    }

    if(found=1) return;
    else
       { if(j=0) group_set[1]=k1;
         else
             { if(k1>group_set[j])
                  group_set[j+1]=k1;
               else
                   {i=1;
                    while(k1>group_set[i])
                         {i=i+1;
                         }
                    for(t=j;t>=i;t--)
                       { group_set[j+1]=group_set[t];
```

```
                              }
                         group_set[i]=k1;
                        }
                  }
          }
}


void add_to_k_set_for_group_set(int k1)
{
       int i=0,j=0,t=0;
       int found;

       i=1;
       j=k_set_for_group_set[0];
       found=0;
       for(i=1;i<=j;i++)
       { if( k1=k_set_for_group_set[i] )
            found=1;
       }

       if(found=1) return;
       else
           { if(j=0) k_set_for_group_set[1]=k1;
             else
                 { if(k1>k_set_for_group_set[j])
                        k_set_for_group_set[j+1]=k1;
                   else
                       {i=1;
                        while(k1>k_set_for_group_set[i])
                              {i=i+1;
                              }
                        for(t=j;t>=i;t--)
                           { k_set_for_group_set[j+1]=k_set_for_group_set[t];
                           }
                        k_set_for_group_set[i]=k1;
                       }
                 }
           }
}


void add_to_z_i(int k1)
{
       int i=0,j=0,t=0;
       int found;

       i=1;
       j=z_i[0];
       found=0;
       for(i=1;i<=j;i++)
       { if( k1=z_i[i] )
            found=1;
       }

       if(found=1) return;
       else
```

```
          { if(j=0) z_i[1]=k1;
            else
                { if(k1>z_i[j])
                     z_i[j+1]=k1;
                  else
                      {i=1;
                       while(k1>z_i[i])
                            {i=i+1;
                            }
                       for(t=j;t>=i;t--)
                          { z_i[j+1]=z_i[t];
                          }
                       z_i[i]=k1;
                      }
                }
          }
}

void check_if_group_set_is_equal_to_z_i(int group_set[I],int z_i[I])
{
      int t=0,j=0;

      result=1;
      if(group_set[0]=z_i[0])
        {
          j=group_set[0];
          for(t=1;t<=j;t++)
             { if(group_set[t]!=z_i[t])
                  result=0;
             }
        }
      else result=0;
}


void prepare_union_of_ifmn_sets_group_set()
{
      int t=0,i2=0,j2=0;
      int k2,i3,j3,i4;
      int found;

      for(t=0;t<=I;t++)
      { union_of_ifmn_sets_group_set[t]=union_of_ifmn_sets[t];
      }

      i2=1;
      j2=group_set[0];
      found=0;

      for(i2=1;i2<=j2;i2++)
          { k2=group_set[i2];
            i3=1;
            j3=union_of_ifmn_sets_group_set[0];
            for(i3=1;i3<=j3;i3++)
               { if(k2=union_of_ifmn_sets_group_set[i3])
                   { for(i4=i3+1;i4<=j3;i4++)
```

```
                           {union_of_ifmn_sets_group_set[i4-
1]=union_of_ifmn_sets_group_set[i4];
                           }

union_of_ifmn_sets_group_set[0]=union_of_ifmn_sets_group_set[0]-1;
                      }
                 }//end of for i3
             }//end of for i2
}


void solution_improvement_using_sets()
{
      int k1=0,i1=0;
      float Benefit_1,Benefit_2,B1_check,B2_check;
     // int z_i[I+1];
    //  union_of_ifmn_sets[I+1];
     // group_set[I+1];
     // k_set_for_group_set[I+1];

     int new_combination_found, new_member_k_is_added,
member_has_common_element;

      union_of_ifmn_sets[0]=0;
      for(i1=1;i1<=I;i1++)
         { union_of_ifmn_sets[i1]=0;
         }

      for(k1=1;k1<=m1;k1++)
         {   for(i1=1;i1<=ifmn[k1][0];i1++)
               {   add_to_union_of_ifmn_sets(ifmn[k1][i1]);
               }
         }

      group_set[0]=0;
      for(i1=1;i1<=I;i1++)
         { union_of_ifmn_sets[i1]=0;
         }
      for(i1=1;i1<=ifmn[1][0];i1++)
         {   add_to_group_set(ifmn[1][i1]);
         }

      k_set_for_group_set[0]=1;
      k_set_for_group_set[1]=1;

      new_member_k_is_added=1;
      { while (new_member_k_is_added=1)
              { new_member_k_is_added=0;
                  {for(k1=2;k1<=m1;k1++)
                      { k1_in_k_set_for_group_set(k1);
                        if(result=0)
                          { member_has_common_element=0;
                            {for(i1=1;i1<=ifmn[k1][0];i1++)
                                {ifmn_in_group_set(ifmn[k1][i1]);
                                 if(result=1)
                                  member_has_common_element=1;
                                }
```

```
                                    if (member_has_common_element=1)
                                      {for(i1=1;i1<=ifmn[k1][0];i1++)
                                          {add_to_group_set(ifmn[k1][i1]);
                                  }
                                         add_to_k_set_for_group_set(k1);
                                         new_member_k_is_added=1;
                              }//end of if member has common element
                          }
                      }//end of result=false
                       }//end of for k1=2
                 }//end of new member k is added=false
              }//end of new member k is added=true
         }//end of while

        Benefit_1=0.0;
        Benefit_2=0.0;
        z_i[0]=0;
        for(i1=1;i1<=I;i1++)
           { z_i[i1]=0;
           }
        { for(i1=1;i1<=p1;i1++)
             {add_to_z_i(i1);
           }
        }

        check_if_group_set_is_equal_to_z_i(group_set,z_i);
        if (result=1)
           { printf("%d\n","The solution can not be improved further using set
heuristic");
             improvement_possible=0;
         }
        else { for(k1=1;k1<=m1;k1++)
                {k_in_k_set_for_group_set(k1);
                  if (result=1)
                     Benefit_1=Benefit_1+d[k1];
                  else Benefit_2=Benefit_2+d[k1];
                  {for(i1=1;i1<=p1;i1++)
                      { i1_in_group_set(i1);
                      if (result=1)
                         Benefit_1=Benefit_1-b[i1];
                      else
                         { prepare_union_of_ifmn_sets_group_set;
                           i1_in_union_of_ifmn_sets_group_set(i1);
                           if(result=1)
                           Benefit_2=Benefit_2-b[i1];
                         }
                      }//end of for i1=1
              }//end of else benefit 2
                }//end of for k1=1
              }//end of else


        new_combination_found=0;
        { for(i1=1;i1<=I;i1++)
              { z_for_increase[i1]=0;
           }
        }
```

```
{ for(k1=1;k1<=K;k1++)
      {k_for_increase[k1]=0;
    }
}
B1_check=0;
B2_check=0;
B1_check=round(100000*Benefit_1);//ROUND
B2_check=round(100000*Benefit_2);//ROUND

if ((Benefit_1>0.0) && (B1_check>0.0))
 { for(i1=1;i1<=p1;i1++)
        { i1_in_group_set(i1);
          if (result=1)
              z_for_increase[i1]=1;
      }

      for(k1=1;k1<=m1;k1++)
        { k1_in_k_set_for_group_set(k1);
          if (result=1)
            k_for_increase[k1]=1;
          new_combination_found=1;
        }
  }//end of if

 else if ((Benefit_2>0.0) && (B2_check>0.0))
   {
    for(i1=1;i1<=p1;i1++)
       { i1_in_union_of_ifmn_sets_group_set(i1);
           if(result=1)
             z_for_increase[i1]=1;
       }

    for(k1=1;k1<=m1;k1++)
       { k1_in_k_set_for_group_set(k1);
           if(result=0)
              k_for_increase[k1]=1;
         }
         new_combination_found=1;

  }// end of else if

  if (new_combination_found=0)
         { improvement_possible=0;
           printf("%d","This Heuristic Using sets terminates here");
         }
  else
       { improvement_possible=1;
         { for(i1=0;i1<=I;i1++)
          { collection_of_zi_that_can_be_increased[i1]=0;
            }
       }
       { for(i1=1;i1<=p1;i1++)
              if (z_for_increase[i1]=1)
         {
collection_of_zi_that_can_be_increased[0]=collection_of_zi_that_can_be_increased
[0]+1;
```

```
collection_of_zi_that_can_be_increased[collection_of_zi_that_can_be_increased[0]
]=i1;
                  }
           }
           }//end of else
} //end of void

/*solution improvement using sets end*/

void try_other_combinations()

{
 int temp,k1,k2,k3,k4,i1,i2,i3,i4;
 float lb_check,Tb_check,local_benefit,total_benefit;
 int sorted_ifmn_k[K+1];
 int combination_found;

       lb_check=0.0;
       Tb_check=0.0;
       for(k1=1;k1<=m1;k1++)
            {sorted_ifmn_k[k1]=k1;
            }

       for(k1=1;k1<=(m1-1);k1++)
           {for(k2=(k1+1);k2<=m1;k2++)
               {if (ifmn[sorted_ifmn_k[k1]][0]<ifmn[sorted_ifmn_k[k2]][0])
                   {
                      temp=sorted_ifmn_k[k1];
                    sorted_ifmn_k[k1]=sorted_ifmn_k[k2];
                    sorted_ifmn_k[k2]=temp;
               }
             }
            }

       k1=0;
       combination_found=false;
       while ((k1<m1) && !(combination_found))
           {
             k1=k1+1;
             for(i1=1;i1<=I;i1++)
                {z_for_increase[i1]=0;
                }

             for(k3=1;k3<=K;k3++)
                  { k_for_increase[k3]=0;
                  }

             k2=sorted_ifmn_k[k1];
             if (ifmn[k2][0]>=2)
                {
                  total_benefit=0;
                  for(i1=1;i1<=ifmn[k2][0];i1++)
                     { total_benefit=total_benefit+(-1)*b[ifmn[k2][i1]];
                       z_for_increase[ifmn[k2][i1]]=1;
                 }
```

```
                    total_benefit=total_benefit+d[k2];
                    k_for_increase[k2]=1;//checked till here
                for(k3=(k1+1);k3<=m1;k3++)
                        { k4=sorted_ifmn_k[k3];
                          local_benefit=d[k4];
                          for(i2=1;i2<=ifmn[k4][0];i2++)
                              { i3=ifmn[k4][i2];
                                if (z_for_increase[i3]=0)
                                   {local_benefit=local_benefit-b[i3];
                          }
                              }

                          lb_check=round(100000*local_benefit);
                          if ((local_benefit>0.0) && (lb_check>0.0))
                              {
                          for(i2=1;i2<=ifmn[k4][0];i2++)
                                 {
                                   i3=ifmn[k4][i2];
                                   z_for_increase[i3]=1;
                                 }

                              k_for_increase[k4]=1;
                              total_benefit=total_benefit+local_benefit;
                              }
                    }//end of for k3
               // }//end of if ifmn[k2][0]>=2

                    Tb_check=round(100000*total_benefit);
                    if ((total_benefit>0.0) && (Tb_check>0.0))
                       {combination_found=true;
                    }
                  }//end of ifmn[k2][0]
             }//end of while
        if (combination_found!=true)
             { //solution_improvement_using_sets();
           }
        else
             { improvement_possible=true;
             for(i1=0;i1<=I;i1++)
                       {collection_of_zi_that_can_be_increased[i1]=0;
                  }

             for(i1=1;i1<=p1;i1++)
                       {if (z_for_increase[i1]=1)

{collection_of_zi_that_can_be_increased[0]=collection_of_zi_that_can_be_increase
d[0]+1;

collection_of_zi_that_can_be_increased[collection_of_zi_that_can_be_increased[0]
]=i1;
                           }
                       }

             }//end of else

}//end of void
```

```
void dual_solution_can_be_improved()

{
 float Benefit_for_single_zi[I];
 int index_no, index_i;
 int k_value,no_in_col_with_min_value;
 float B_check;

          B_check=0.0;
          improvement_possible=false;
          for(index_i=1;index_i<=p1;index_i++)
             {Benefit_for_single_zi[index_i]=-1*b[index_i];
             }

          index_i=0;
           while (((Benefit_for_single_zi[index_i]>0) && (B_check>0.0)) ||
(index_i=p1))
          {     index_i=index_i+1;
              //    printf(" dual_solution_can_be_improved entering in min in
col\n");

                no_in_col_with_min_value=min_in_col[0][index_i];
             for(index_no=1;index_no<=no_in_col_with_min_value;index_no++)
                       {
                        k_value=min_in_col[index_no][index_i];
                        if (ifmn[k_value][0]=1)

{Benefit_for_single_zi[index_i]=Benefit_for_single_zi[index_i]+d[k_value];
                       }
                 }
                B_check=round(100000*Benefit_for_single_zi[index_i]);
          }
        // while (((Benefit_for_single_zi[index_i]>0) && (B_check>0.0)) ||
(index_i=p1));

         B_check=0.0;
         B_check=round(100000*Benefit_for_single_zi[index_i]);
         if ((Benefit_for_single_zi[index_i]>0) && (B_check>0.0))
            {
               improvement_possible=true;
               collection_of_zi_that_can_be_increased[0]=1;
               collection_of_zi_that_can_be_increased[1]=index_i;
            }
//          printf(" dual_solution_can_be_improved entering in try other
combinations\n");

         else try_other_combinations();
printf(" dual_solution_can_be_improved complted\n");
}//end of void


void determine_extent_of_increase()
{
  int zr,i2,temp;
  float zivr[K+1];
  int ks_for_positive_zivr[K+1];
```

```
float B_check,Benefit;

    B_check=0.0;
    extent_of_increase=0;
    ks_for_positive_zivr[0]=0;
    for(k1=1;k1<=m1;k1++)
         { ks_for_positive_zivr[k1]=0;
           zivr[k1]=0;
      }

    if (collection_of_zi_that_can_be_increased[0]==1)
      {
          zr=collection_of_zi_that_can_be_increased[1];
        for(k1=1;k1<=m1;k1++)
            { if ((ifmn[k1][0]=1) && (ifmn[k1][1]=zr))
               {
                     zivr[k1]=B2K[k1]-B1K[k1];
                     ks_for_positive_zivr[0]=ks_for_positive_zivr[0]+1;
                     ks_for_positive_zivr[ks_for_positive_zivr[0]]=k1;
                   }
            }

     }
    else
      {
        for(k1=1;k1<=K;k1++)
              { B3K[k1]=0;
            }

        for(k1=1;k1<=m1;k1++)
              { B3K[k1]=1000000000;
              for(i1=1;i1<=p1;i1++)
                     {if (z_for_increase[i1]=0)
                         { if (BIK_plus_zi[k1][i1]<B3K[k1])
                      {B3K[k1]=BIK_plus_zi[k1][i1];
                        }
                     }
                 if (k_for_increase[k1]=1)
                         { zivr[k1]=B3K[k1]-B1K[k1];
                           ks_for_positive_zivr[0]=ks_for_positive_zivr[0]+1;
                           ks_for_positive_zivr[ks_for_positive_zivr[0]]=k1;
                         }
                     }//end of for on i1


              }//end of for k1
     }//end of else

     for(i1=1;i1<=(ks_for_positive_zivr[0]-1);i1++)
          { for( k1=i1+1; k1<=ks_for_positive_zivr[0];k1++)
                { if
(zivr[ks_for_positive_zivr[i1]]>zivr[ks_for_positive_zivr[k1]])
                     {
                        temp=ks_for_positive_zivr[i1],
                        ks_for_positive_zivr[i1]=ks_for_positive_zivr[k1];
                        ks_for_positive_zivr[k1]=temp;
                     }
```

```
                              }
                  }


        temp=collection_of_zi_that_can_be_increased[0];
        Benefit=0;
        for(i1=1;i1<=temp;i1++)
             {
               i2= collection_of_zi_that_can_be_increased[i1];
               Benefit=Benefit-b[i2];
             }

        for(k1=1;k1<=ks_for_positive_zivr[0];k1++)
             { Benefit=Benefit+d[ks_for_positive_zivr[k1]];
           }

        k1=0;
        B_check=round(100000*Benefit);
        while ((Benefit>=0) && (B_check>0.0) && (k1<=ks_for_positive_zivr[0]))
               {
                        k1=k1+1;
                        Benefit=Benefit-d[ks_for_positive_zivr[k1]];
             }
        extent_of_increase=zivr[ks_for_positive_zivr[k1]];
        for(i1=1;i1<=temp;i1++)
             {
               i2=collection_of_zi_that_can_be_increased[i1];
               value_of_zi[i2]=value_of_zi[i2]+extent_of_increase;
             }

  }//end of void


  void improve_the_solution()

  {
    int i2,k2;

    current_loss=0;
    for(i2=1;i2<=collection_of_zi_that_can_be_increased[0];i2++)
        { current_loss=current_loss-
b[collection_of_zi_that_can_be_increased[i2]]*extent_of_increase;

          for(k2=1;k2<=m1;k2++)

{BIK_plus_zi[k2][collection_of_zi_that_can_be_increased[i2]]=BIK_plus_zi[k2][col
lection_of_zi_that_can_be_increased[i2]]+extent_of_increase;
          }//end of for on k2
        }//end of for on i2

  }


  void compute_the_improved_solution()

  {
```

```
 int k2;

 objective_fn_value=0;
 for(k2=1;k2<=m1;k2++)
     { objective_fn_value=objective_fn_value+B1K[k2]*d[k2];
     }

 cumulative_loss=cumulative_loss+current_loss;
 objective_fn_value=objective_fn_value+cumulative_loss;
}


void dual_var_calculation()

{
 float larg,obj;
 for(k1=1;k1<=m1;k1++)
      { value_of_vk[k1]=0;
      }

 v0=0;
 larg=0;
 for(k1=1;k1<=m1;k1++)
      {if (B1K[k1]>larg)
          {larg=B1K[k1];
          }
      }

 v0=larg;
 obj=0;
 for(k1=1;k1<=m1;k1++)
      {
          value_of_vk[k1]=v0-B1K[k1];
          obj=obj-value_of_vk[k1]*d[k1];
      }

 obj_fn_value_using_v0_vk=v0+obj+cumulative_loss;

}//end of void


void print_solution()
{
      int c1;


printf("********************************************************************\n\n
");
      printf("The final cost matrix is given below\n\n");

      printf("------------------------------------------------------------");
      for(k1=1;k1<=m1;k1++)
          {
            for(i1=1;i1<=p1;i1++)
                {
                 printf("%d\t",BIK_plus_zi[k1][i1]);
                }
```

```
            printf("\n");
          }
        printf("------------------------------------------------------------
\n");

/*
        printf("The minimum cost in each row is\n");
        c1=0;
        for(k1=1;k1<=m1;k1++)
         {
           c1=c1+1;
           write(B1K[k1]:9:2);
              if (c1>13)
             {
                printf("\n");
                c1=0;
             }
        }
        printf("\n");
        printf("------------------------------------------------------------
\n");

        printf("The values of DUAL VARIABLES are given as\n");
        printf("------------------------------------------------------------
\n");

        printf('v0=',v0);
        printf("\n");
        printf("The values of vks are given as\n");

        c1=0;
        for(k1=1;k1<=m1;k1++)
        {
         c1=c1+1;
         write(value_of_vk[k1]:9:2);
              if (c1>13)
             {
                printf("\n");
                c1=0;
             }
        }
        printf("\n\n");

        c1=0;
        printf("The values of zis are given as\n");

        for(i1=1;i1<=p1;i1++)
        {
         c1=c1+1;
         write(value_of_zi[i1]:9:2);
              if (c1>13)
             {
               printf("\n");
               c1=0;
               }
        }
        printf("\n\n");
```

```
*/
        printf("--------------------------------------------------------------
\n");


printf("********************************************************************\n")
;
        printf("\nNO_OF_ITERATION=%d\n",No_of_iteration);
        printf("\nOPTIMUM OBJ. FUNCTION VALUE=%d\n",objective_fn_value);
        printf("\nOPT. OBJ. FUNC. VALUE USING V0 and
VK=%d\n",obj_fn_value_using_v0_vk);
        printf("\n");

printf("********************************************************************\n\n
");

}



main()
{
        initialization();
        printf("cumulative_loss= %f\n",cumulative_loss);
        printf("current_loss= %f\n",current_loss);
        printf("extent_of_increase= %f\n",extent_of_increase);

        read_input();

        prepare_set_ifmn_k();

        prepare_min_in_col_array();

        obtain_solution_with_all_zi_at_zero();

        dual_solution_can_be_improved();

        No_of_iteration=0;
        while (improvement_possible)
        {
          No_of_iteration=No_of_iteration+1;
          determine_extent_of_increase();

          improve_the_solution();

          prepare_set_ifmn_k();

          prepare_min_in_col_array();

          compute_the_improved_solution();

        dual_solution_can_be_improved();
        }
        dual_var_calculation();

        print_solution();
/*
```

```
        printf("\n");
        printf("The values of bis are");
        c2=0;
        for(i1=1;i1<=p1;i1++)
        {
         c2=c2+1;
         write(b[i1]:10:6);
              if (c2>13)
            {
               printf("\n");
               c2=0;
            }
       }
      printf("\n\n");
      c2=0;
      printf("The values of dks are ");
      for(k1=1;k1<=m1;k1++)
      {
       c2=c2+1;
       write(d[k1]:10:6);
             if (c2>13)
           {
              printf("\n");
              c2=0;
           }
      }
      printf("\n\n");
      printf("The value of ifmns with first value showing count of ifmns are as
follow\n");

      for(k1=1;k1<=m1;k1++)
      {
       c2=0;
       for(i1=0;i1<=p1;i1++)
      {
       c2=c2+1;
       write(ifmn[k1,i1]:4);
             if (c2>35)
           {
              printf("\n");
              c2=0;
             }
      }
      printf("\n");
*/
}
```